KERKLOH
TORNSDORF
ZOLLER

Das große Buch zu Amiga DOS

DATA BECKER

AMIGA

Kickstart 1.3

Kerkloh
Tornsdorf
Zoller

Das große Buch zu AmigaDOS

DATA BECKER

1. Auflage 1988

ISBN 3-89011-306-0

Copyright © 1988

DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Text verarbeitet mit Word 4.0, Microsoft
Ausgedruckt mit Hewlett Packard LaserJet II
Druck und Verarbeitung Mohndruck, Gütersloh

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort zur Neuauflage

In wenigen Tagen wird jedem Amiga eine vollständig überarbeitete Version der Workbench-Diskette beiliegen. Besitzer eines Amiga 1000, dessen Betriebssystem ja im Gegensatz zu dem des Amiga 500 bzw. 1000 nicht fest in einem Speicherbaustein eingebrannt ist, können sich sogar über ein überarbeitetes Kickstart freuen.

Diese neuen Systemdisketten mit der Versionsnummer 1.3 bieten so viele Neuerungen und Verbesserungen, daß wir es für notwendig hielten, sie noch vor ihrem offiziellen Erscheinungstermin in Deutschland mit in die Neuauflage aufzunehmen.

Die Autoren *Münster, im Juni 1988*

Vorwort

Der Amiga ermöglicht durch seine grafische Benutzeroberfläche und die Maussteuerung gerade dem Neuling einen einfachen Einstieg in die Welt der Computer. Die kleinen Bilder und Symbole, die automatisch nach dem Start des Rechners auf dem Bildschirm erscheinen, sehen ja auch wirklich etwas einladender aus als ein sturer Cursor, der oben links in der Ecke sitzt und auf Tastatureingaben wartet.

Früher oder später passiert es aber jedem, daß er aus Versehen oder aus reiner Neugierde das CLI-Symbol auf der Workbench-Diskette anklickt. Es erscheint ein Fenster, und prompt meldet sich auch der langweilige Cursor. Nun hilft auch keine Maus mehr, ein Schließsymbol besitzt das Fenster nicht. Man muß sich wohl oder übel ein bißchen mit dem CLI auseinandersetzen,

wenn man das Fenster nicht mittels des Netzschalters des Amiga schließen will.

Vielleicht haben Sie aber auch schon längst gemerkt, daß man für eine sinnvolle Arbeit mit dem Amiga ohne das CLI gar nicht auskommen kann. Die Workbench ist in Wirklichkeit doch nur eine Fassade, die den Einblick auf das Wesentliche verhindert.

In jedem Fall wird Ihnen dieses Buch bei der Arbeit im CLI des Amiga hilfreich zur Seite stehen. Nach einer einfach gehaltenen Einführung finden Sie jede Menge Informationen: Lösungen für die alltäglichen kleinen Schwierigkeiten im CLI, ausführliche Behandlung aller CLI-Befehle, Programmierung von Batch-Dateien, Informationen über Multitasking, Handler, Devices und noch vieles mehr. Auf den letzten Seiten rundet die Quick-Reference zum schnellen Nachschlagen aller Befehle das Gesamtkonzept ab.

Wir wünschen Ihnen mit diesem Buch viel Spaß und Erfolg bei der spannenden Arbeit im Command Line Interface Ihres Amiga.

Die Autoren *Münster, im November 1987*

Inhaltsverzeichnis

1.	**Einführung**	**15**
1.1	Die Aufgabe des DOS	15
1.2	Über die Workbench und das CLI	16
1.3	Vorbereitungen	17
1.4	Das CLI meldet sich	19
1.5	Erste Erfolge stellen sich ein	20
1.6	Die Verzeichnisstruktur	22
1.7	Die Help-Funktion	25
1.8	Beenden der Arbeit im CLI	28
2.	**Die CLI-Befehle und ihre Aufgaben**	**29**
2.1	Befehle zur Laufwerks- und Dateiverwaltung	30
2.1.1	Format	30
2.1.2	Dir	34
2.1.3	CD	38
2.1.4	MakeDir	40
2.1.5	Delete	42
2.1.6	Copy	44
2.1.7	List	48
2.1.8	Rename	56
2.1.9	DiskCopy	58
2.1.10	ReLabel	60
2.1.11	Info	61
2.1.12	Install	63
2.1.13	Type	68
2.1.14	Join	70
2.1.15	Search	71
2.1.16	Sort	73
2.1.17	Protect	74
2.1.18	FileNote	78
2.1.19	SetDate	79
2.1.20	DiskDoctor	80
2.1.21	DiskChange	86

2.2 Systemkommandos des CLI 86
2.2.1 NewCLI 87
2.2.2 EndCLI 93
2.2.3 Run 93
2.2.4 Status 95
2.2.5 ChangeTaskPri 97
2.2.6 Break 99
2.2.7 Path 100
2.2.8 Assign 102
2.2.9 AddBuffers 105
2.2.10 Why 106
2.2.11 Fault 107
2.2.12 Date 108
2.2.13 SetClock 109
2.2.14 Prompt 110
2.2.15 Stack 111
2.2.16 BindDrivers 112
2.2.17 Mount 113
2.3 Befehle für Batch-Dateien 116
2.3.1 Execute 117
2.3.2 Echo 120
2.3.3 Failat 121
2.3.4 Quit 123
2.3.5 If/Else/EndIf 123
2.3.6 Ask 127
2.3.7 Skip/Lab 128
2.3.8 Wait 129
2.3.9 Version 131
2.4 Die Editoren ED und Edit 132
2.4.1 ED 133
2.4.2 Edit 137

3. Die Standard-Devices des Amiga 147
3.1 Die Floppy-Devices 147
3.2 Das RAM-Disk-Device 148
3.3 Das parallele Device 150
3.4 Das serielle Device 151
3.5 Das Drucker-Device 153
3.6 Das Con-Device 154

4.	**Die neue Workbench 1.3**	**157**
4.1	Neue CLI-Befehle	158
4.1.1	Avail	158
4.1.2	FF	159
4.1.3	Lock	160
4.1.4	NewShell	160
4.1.5	Remrad	164
4.1.6	Resident	164
4.1.7	SetAlert	168
4.1.8	Setenv/Getenv	169
4.1.9	XIcon	169
4.2	Neue Devices	170
4.2.1	Das NewCon-Device	170
4.2.2	Das RAD-Devive	171
4.2.3	Da Pipe-Device	172
4.2.4	Das Speak-Device	173
4.2.5	Das Aux-Device	174
4.2.6	Das Fast-File-System	174
5.	**Hinweise zur Arbeit mit dem CLI**	**177**
5.1	Über Ein- und Ausgaben im CLI	177
5.2	Die Jokerzeichen	178
5.3	Unterbrechungen im CLI	180
5.4	Die RAM-Disk und das CLI	183
5.5	Drucken vom CLI aus	188
5.5.1	File-Ausdruck mit Copy	188
5.5.2	Allgemeines Umlenken der Ausgabe	190
5.5.3	Steuerzeichen für den Drucker	191
5.6	Anwendungen zum Con-Device	193
5.7	Anwendungen zur seriellen Schnittstelle	197
6.	**Batchprocessing auf dem Amiga**	**205**
6.1	Einführung in die Stapeldateiverarbeitung	205
6.1.1	Was sind Batch-Files?	205
6.1.2	Wie sehen Batch-Files aus?	206
6.1.3	Wie werden Batch-Files aufgerufen?	207
6.1.4	Ein einfaches Beispiel	207

6.2	Modifikationen an der Startup-Sequence der Workbench 1.2	209
6.2.1	Komfortable Startup-Sequence der Workbench 1.2	216
6.2.2	Die Startup-Sequencen der Workbench 1.3	219
6.3	Nützliche Batches	221
6.3.1	Eine besondere Drucker-Batch-Datei	222
6.3.2	Eigene Batch-Befehle erstellen	225
6.3.3	Die Arbeits-Workbench 1.3	229
6.3.4	Starten von Batch-Dateien mit der Maus	236
6.3.5	Batch-Dateien "typen"	239
6.3.6	Alles in die RAM-Disk	240
6.4	Neue Shell-Befehle mit Alias	241
7.	**AmigaDOS und Multitasking**	**247**
7.1	Was ist Multitasking?	248
7.2	Multitasking mit CLI und Workbench	250
7.3	Multitasking mit NewCLI	254
7.4	Multitasking mit Run	257
7.5	Wie man das CLI am besten einsetzt	262
7.6	Tiefer Eingriff mit ChangeTaskPri	265
7.7	Was zu beachten ist	269
8.	**AmigaDOS Intern**	**275**
8.1	DOS, Devices, Handler, Packets	275
8.2	Relokatible Programme, Segmente, BPTR-Pointer..	279
8.3	Das Programm CLI	282
8.4	Die interne DOS-Bibliothek	284
8.5	Das Beispiel Run	288
9.	**Neue CLI-Befehle**	**293**
9.1	Die Realisierung in C	295
9.2	Task - Die Aufgaben des Amiga	299
9.3	TaskPri - Die Aufgabenverteilung beliebig ändern	302
9.4	TaskStop - Nichts geht mehr	307

9.5 Delay - Tasks dynamisch verzögern 310
9.6 Beliebige Zeichenketten in Dateien ersetzen - Ersetze 314

10. Quick-Reference für ED, Edit & CLI-Kommandos 325

10.1 Die ED-Funktionen 325
10.2 Die Edit-Funktionen 328
10.3 Die CLI-/Shell-Befehle 336
10.4 Die Befehle alpabetisch geordnet 354

Anhang **361**

Index **363**

1. Einführung

Die folgenden Abschnitte sollen Ihnen den Einstieg in die Arbeit mit dem CLI so leicht wie möglich machen. Aus diesem Grund wird am Anfang die Theorie auf ein absolut notwendiges Maß beschränkt. Für die Anwendung des CLI reicht diese Einführung aber völlig aus; wer darüber hinaus noch genauer in die Materie einsteigen will, findet im Kapitel 6 eine Fülle von Hintergrund-Informationen rund um das CLI.

1.1 Die Aufgabe des DOS

Nein, in der Überschrift ist kein Druckfehler. Bevor wir uns mit dem eigentlichen CLI beschäftigen können, müssen wir uns ein wenig mit den Aufgaben des sogenannten DOS auseinandersetzen. DOS ist die Abkürzung für "Disk Operating System", was frei übersetzt etwa soviel heißt wie "laufwerkorientiertes Betriebssystem". Unter einem Betriebssystem versteht man allgemein die Gesamtheit aller Programme, die den Betrieb eines Computers erst ermöglichen. Man darf dies nicht mit Anwendungsprogrammen (Textverabeitung, Kalkulation etc.) verwechseln. Ein Betriebssystem liefert nur das Fundament, auf das der Anwender mit seinen Programmen aufbauen kann. Es übernimmt Aufgaben wie Speicherverwaltung, Ansteuerung der Hardware (Tastatur, Bildschirm, Floppy, Drucker etc.) und die Koordination der verschiedenen Aufgaben. Es stellt auch fertige Programmfunktionen zur Verfügung. Man muß sich als Systemprogrammierer zum Beipiel nicht darum kümmern, welche Speicherbereiche im Rechner schon belegt sind und welche man noch für eigene Anwendungen benutzen kann. Das Betriebssystem teilt einem automatisch einen freien Platz der gewünschten Größe zu.

Die Besonderheit des DOS liegt nun darin, daß die Befehle, die der Rechner ausführen kann, nicht mit in das eigentliche Betriebssystem integriert sind. Bei den kleineren Homecomputern kann man bestimmte Befehle eingeben (z.B. Load, Save, Run

usw.), die das Betriebssystem selbst erkennt und sofort ausführt. Das AmigaDOS basiert auf einem etwas anderen Prinzip: Die Befehle sind kleine Programme, die erst bei Bedarf von einem Laufwerk (Floppy, Festplatte oder RAM-Disk) geladen werden und bei ihrer Ausführung auf die Routinen des im Speicher befindlichen Betriebssystems zurückgreifen. Dieses Verfahren weist einige Vorteile gegenüber den Betriebssystemen auf, die ihre Befehle integriert haben:

1. Jeder Befehl belegt nur solange kostbaren Speicherplatz im Rechner, bis er seine Funktion beendet hat. Er wird anschließend sofort wieder gelöscht.
2. Sollte sich mit der Zeit herausstellen, daß ein Befehl noch Fehler enthält, kann er auch später noch sehr leicht gegen eine korrigierte Version ausgetauscht werden.
3. Der Befehlsumfang ist nicht beschränkt. Jederzeit kann das DOS um neue Befehle ergänzt werden.

Nicht verschwiegen werden darf jedoch der große Nachteil, daß besonders bei kleinen Rechnerkonfigurationen mit wenig Speicherplatz und nur einem Laufwerk der Bedienungskomfort leidet, da hier häufiger die Disketten gewechselt werden müssen und viel Zeit mit den Ladevorgängen der einzelnen Befehle vergeht. Bei Verwendung einer Festplatte und intensiver Nutzung der RAM-Disk fällt auch dieser Nachteil weg.

1.2 Über die Workbench und das CLI

Die Aufgaben des DOS haben wir oben schon grob abgesteckt. Was noch fehlt, sind die Werkzeuge, mit denen der Anwender die Abarbeitung bestimmter Befehle veranlassen kann. Konkret: Wie können wir dem Rechner mitteilen, daß wir jetzt z.B. eine neue Diskette formatieren möchten? Natürlich kann dies über die altbekannte Workbench geschehen: In dem Menü mit dem Namen Disk gibt es einen Unterpunkt Initialize. Hat man vorher das gewünschte Diskettensymbol angeklickt und wählt nun diesen Punkt an, wird der entsprechende Befehl von der Workbench-Diskette geladen und alles weitere veranlaßt. In Wirklichkeit ist die Workbench nichts anderes als ein kleines Programm,

das ganz zu Beginn von der Diskette geladen wird und die grafische Aufbereitung der Benutzeroberfläche bewirkt.

Die Alternative zur Workbench stellt das CLI dar. Diese drei Buchstaben stehen für Command Line Interface, was etwa soviel bedeutet wie Kommandozeilen-Schnittstelle. Der Namen verrät eigentlich schon alles: Die Befehle werden hier nicht mehr wie in der Workbench durch das Anklicken der Symbole aktiviert, sondern durch ihren Aufruf mittels Tastatureingaben. Das CLI kommt daher auch mit einem nüchternen Fenster aus. Mit der Maus kann man dann nur noch bei Bedarf die Ausmaße dieses CLI-Fensters variieren.

Ist das CLI denn dann kein Rückschritt? Vielleicht mag das auf den ersten Blick so aussehen, denn die Workbench vereinfacht gerade Einsteigern die Arbeit mit dem Computer ungemein. Wenn man tatsächlich nur ab und zu seinen Amiga einschaltet, um mit einem bestimmten Textverarbeitungsprogramm zu arbeiten, kann man sogar auf das CLI ganz verzichten. Wer jedoch über das Einsteigerstadium hinauskommen will (und wer will das nicht?), wird um das CLI nicht herumkommen. Einige Probleme lassen sich nämlich von der Workbench aus nur über Umwege oder gar nicht lösen. Beispielsweise kann nur vom CLI aus die sogenannte Startup-Sequence verändert werden. Hierbei handelt es sich um eine spezielle Befehlsdatei, die sich auf der Workbench-Diskette befindet und bei jedem Start des Amiga abgearbeitet wird. Es werden nämlich noch längst nicht alle Dateien, die sich auf einer Diskette befinden, auf der Workbench auch tatsächlich angezeigt. Das CLI ist somit die einzige Möglichkeit, um im Amiga wirklich hinter die Kulissen schauen zu können.

1.3 Vorbereitungen

Bevor Sie beginnen, mit dem CLI zu arbeiten, sollten Sie unbedingt eine Kopie Ihrer originalen Workbench-Diskette anfertigen. Mit der Zeit werden Sie nämlich feststellen, daß diese oder jene Programme für Sie überflüssig sind und gelöscht werden können. Sollten Sie aber irgendwann einmal doch wieder solch ein Programm benötigen, können Sie auf die unveränderte Ori-

ginaldiskette zurückgreifen. Außerdem kann es immer einmal passieren, daß wichtige Daten aufgrund einer defekten Diskette oder unbeabsichtigten Löschens verlorengehen.

Von der Workbench aus ist die Herstellung der Kopie kein Problem. Wenn Sie dabei noch Schwierigkeiten haben, können Sie sich an die folgenden Anweisungen halten:

1. Schieben Sie zunächst den kleinen Schreibschutz-Clip der originalen Workbench-Diskette nach außen, falls er sich nicht sowieso schon in dieser Position befindet. Hierdurch wird eine Öffnung freigegeben und ein versehentliches Überschreiben verhindert. Das Verfahren erinnert an die Plastikzungen auf der Rückseite von Musik-Kassetten.

2. Legen Sie diese Diskette in das eingebaute Laufwerk df0:, und starten Sie das System durch gleichzeitiges Drücken der Ctrl-Taste zusammen mit den beiden Tasten links und rechts neben der Leertaste. Sobald die Tasten losgelassen werden, beginnt der Ladevorgang.

3. Nach einer Weile erscheint das gewohnte Bild der Workbench. Die eingelegte Workbench-Diskette wird durch ein Symbol oben rechts dargestellt. Klicken Sie nun dieses Symbol mit der linken Maustaste einmal an, und wählen Sie unter dem Menü Workbench den Punkt Duplicate.

4. Jetzt werden Sie aufgefordert, die FROM-Disk, also die zu kopierende Diskette einzulegen. Natürlich ist das überflüssig, da die Diskette schon längst im Laufwerk liegt.

5. Nun müssen während des Kopiervorgangs mehrere Male abwechselnd Quell- und Zieldiskette miteinander vertauscht werden. Der Schreibschutz-Clip der Zieldiskette muß natürlich die Öffnung verdecken, damit diese beschrieben werden kann.

6. Zur Unterscheidung von der Originaldiskette wird dem Namen der Kopie noch der Text Copy of vorangestellt. Wenn Ihnen dies nicht gefällt, können Sie den Zusatz durch Rename entfernen. Das DOS kann beide Disketten immer noch an ihrem internen Datum und der Uhrzeit

auseinanderhalten. Diese Angaben werden nämlich auf einer Kopie immer aktualisiert abgespeichert.

1.4 Das CLI meldet sich

Starten Sie nun mit der soeben angefertigten Kopie Ihren Amiga. Das ist leider noch einmal notwendig, da das DOS sonst bei jeder Aktion nach der Originaldiskette fragen würde. Aber diese haben Sie ja hoffentlich schon längst wieder eingeschlossen.

Wählen Sie nun auf dem Diskettenordner den Punkt System an. In dem nun erscheinenden Ordner sollte sich auch das Symbol des CLI befinden. Falls dies zutrifft, können Sie die folgenden beiden Absätze überspringen.

Die Entwickler des Amiga haben sich eine kleine Sicherung einfallen lassen. Ist sie eingeschaltet, wird das CLI-Symbol auf der Workbench nicht angezeigt und so ein Start des Programms verhindert. Da aber während der Arbeit mit diesem Buch nichts passieren kann, schalten wir die Sicherung einfach ab. Laden Sie dazu nach dem Schließen des System-Fensters das Programm Preferences. Nach kurzer Zeit erscheint ein großes Fenster. Unter vielen anderen Einträgen findet man hier auf der linken Seite auch ein Feld mit dem Namen CLI. Klicken Sie nun einfach mit der Maus auf das direkt danebenstehende On-Feld und anschließend auf das Feld Save ganz rechts im Bild. Hierdurch wird die neue Systemkonfiguration auf der Diskette abgespeichert. Wenn Sie nun wieder den System-Ordner öffnen, befindet sich dort auch bei Ihnen das CLI-Icon.

Das CLI wird von der Workbench aus durch zweimaliges Anklicken des gleichnamigen Icons geladen und gestartet. Es erscheint dann ein Fenster mit dem Titel New CLI Window.

Nun können Sie auch die beiden Ordner im Hintergrund wieder schließen, da sie für die weitere Arbeit nicht mehr benötigt werden. Das CLI-Fenster besitzt fast alle typischen Eigenschaften, die Fenster im Amiga aufweisen können: Es kann beliebig plaziert und in der Größe verändert werden. Außerdem sind auch die beiden Vordergrund-/Hintergrund-Gadgets vorhanden.

Nur das Schließsymbol oben links in der Ecke fehlt. Selbst hierfür gibt es nämlich schon einen Tastaturbefehl, der aber erst später besprochen wird.

Das einzige, was man im CLI-Fenster sieht, ist das sogenannte DOS-Prompt. Darunter versteht man das oder die Zeichen, mit dem der Computer dem Benutzer zu verstehen gibt, daß er bereit ist, neue Befehle über die Tastatur entgegenzunehmen und auszuführen. Bei unserem CLI besteht also dieses Prompt aus einer Eins und der spitzen Klammer. Daneben wartet nun der Cursor auf unsere Eingaben. Dann kann es ja jetzt endlich losgehen.

1.5 Erste Erfolge stellen sich ein

Alle Eingaben im CLI müssen durch Betätigung der Enter- oder Return-Taste (die Taste mit dem abgewinkelten Pfeil) abgeschlossen werden. Da die beiden Tasten völlig gleichwertig sind, wird zur Vereinfachung im weiteren Verlauf die Enter-Taste nicht mehr erwähnt.

Wird kein Befehl eingegeben, erscheint nach einem Return das Prompt nur eine Zeile tiefer. Leider ist es nicht möglich, mit den vier Pfeiltasten den Cursor auf eine beliebige Position innerhalb des Fensters zu bringen. Daher kann zum Beispiel ein schon einmal eingegebener Befehl, der weiter oben steht, nicht durch ein einfaches Return wiederholt werden. Alle Befehle müssen jedesmal vollständig ausgeschrieben werden. Selbst in der Eingabezeile können einmal eingegebene Zeichen immer nur von rechts nach links nacheinander mit dem Backspace (das ist die Taste links neben Del) gelöscht werden. Ein vollständiges Löschen der aktuellen Zeile bis zum Prompt erzielt man durch gleichzeitiges Drücken der Ctrl-Taste und X.

Natürlich können nur ganz bestimmte Befehle auch wirklich ausgeführt werden. Die Eingabe des Befehls:

```
Inhaltsverzeichnis!
```

wird das CLI beantworten durch:

```
Unknown command Inhaltsverzeichnis!
```

Welche Befehle kennt denn nun das CLI? Die Frage ist eigentlich falsch gestellt. Wie ganz zu Beginn schon erwähnt wurde, können nur solche Befehle bearbeitet werden, die als Programme auf einem Laufwerk vorliegen. Dies war ja gerade die Besonderheit des AmigaDOS. Das CLI hat (vereinfacht) erst einmal nur die Aufgabe, einen Programmnamen entgegenzunehmen, ein gleichnamiges File auf dem Laufwerk zu suchen und dieses dann im Rechnerspeicher zur Ausführung zu bringen. Dabei wird auch gleich deutlich, daß nicht nur typische DOS-Befehle ausgeführt werden können: Jedes lauffähige Programm kann vom CLI aus durch die Angabe seines Namens geladen und gestartet werden.

Die Frage müßte also richtig lauten: Welche DOS-Befehle befinden sich auf der Workbench-Diskette, die man vom CLI aus aufrufen kann?

An dieser Stelle eine lange Liste aller Befehle zu bringen, wäre sicher nicht sehr effektiv. Wir beginnen daher mit einem relativ einfachen, aber dennoch wichtigen Befehl. Sein Name ist Dir (von Directory), und er bewirkt die Ausgabe des Inhaltsverzeichnisses eines bestimmten Laufwerkes (Diskette, Harddisk oder RAM-Disk). Ob Sie im CLI Groß- oder Kleinschreibung wählen, bleibt ganz Ihnen überlassen. Sogar eine gemischte Schreibweise ist bei allen Befehlen zulässig. Bevor Sie diesen Befehl eingeben, sollten Sie das CLI-Fenster auf maximale Größe bringen, da die Ausgabe recht umfangreich wird. Starten Sie nun die Ausgabe durch das Kommando Dir (Return nicht vergessen!).

Nach einer Weile wird nach und nach das Inhaltsverzeichnis der Diskette ausgegeben, die sich im Laufwerk Null befindet. Es handelt sich also um das Verzeichnis der Workbench-Diskette. Besonders schnell kommen die Namen ja wirklich nicht auf den Bildschirm. Daran ist die Methode schuld, mit der die Einträge auf der Diskette verwaltet werden. Falls man das CLI-Fenster nicht groß genug gewählt hat, kann es bei solchen Ausgaben vorkommen, daß nicht alles auf einmal auf den Bildschirm paßt. Die Ausgabe hält jedoch nicht selbsttätig an, wenn eine Zeile

den oberen Fensterrand erreicht. In diesem Fall genügt die Betätigung irgendeiner Taste, um die Ausgabe zu stoppen. Durch Backspace kann die Ausgabe jederzeit wieder fortgesetzt werden.

1.6 Die Verzeichnisstruktur

Einige Namen in der Liste, die der Dir-Befehl ausgibt, werden Ihnen sicher noch von der Workbench bekannt vorkommen, andere wiederum tauchen hier zum ersten Mal auf. Auffällig ist, daß alle Einträge, die in der Workbench zu sehen sind, in dieser Liste gleich zweimal vertreten sind: Einmal unter ihrem einfachen Namen (z.B. System) und ein weiteres Mal mit dem Zusatz .info. Hier haben wir gleich den Grund gefunden, warum die Workbench einige Einträge einfach nicht anzeigen kann: Sie besitzen kein solches .info-File. Es handelt sich dabei um eine kleine Datei, die unter anderem auch die Informationen über das Aussehen des Bildchens (Icons) enthält, mit der die gleichnamige Datei (diesmal jedoch ohne den Zusatz .info im Namen) auf der Workbench dargestellt wird. Für die Arbeit im CLI sind diese Einträge aber überflüssig. Das Programm Preferences zum Beispiel ist auch ohne diese .info-Datei vollkommen lauffähig.

Weiterhin fällt bei der Auswertung der Liste auf, daß sämtliche Einträge, die in der Workbench durch eine Schublade dargestellt werden, beim Dir-Befehl mit dem Kommentar Dir versehen werden. Die Struktur, mit der das AmigaDOS die Dateien verwaltet, ist natürlich bei der Workbench und im CLI gleich. Man bekommt also auch im CLI nicht sofort alle Dateien zu sehen. Es wird zunächst nur das Hauptverzeichnis einer Diskette ausgegeben. Häufig wird bei dieser Art der Dateiverwaltung von einer Baumstruktur gesprochen. Das Hauptverzeichnis stellt den Stamm dar, von dem die Unterverzeichnisse in Form von Ästen ausgehen. Jedes Unterverzeichnis kann entweder richtige Dateien enthalten oder aber wiederum neue Ordner, die weiterführende Äste darstellen. Der Schachtelungstiefe von Unterverzeichnissen sind praktisch keine Grenzen gesetzt.

Wie gelangt man in andere Unterverzeichnisse? Von der Workbench aus ist das kein Problem: Ein Unterverzeichnis wird an-

gezeigt, sobald man eine Schublade zweimal anklickt. Tauchen in diesem neuen Unterverzeichnis weitere Schubladen auf, kann auf die gleiche Weise auch deren Inhalt ausgegeben werden und so weiter.

Möchte man sich vom CLI aus den Inhalt eines ganz bestimmten Unterverzeichnisses ansehen, muß der Dir-Befehl noch um die sogenannte Pfadangabe erweitert werden. Dabei handelt es sich um einen Zusatz, der den Weg beschreibt, wie man vom Stammverzeichnis einer Disk zu dem gewünschten Ordner gelangt. Die Beschreibung dieses Weges erfolgt - wie sollte es anders sein - anhand der Ordnernamen, die zu dem gewünschten Verzeichnis führen. Im einfachsten Fall genügt es also, den Dir-Befehl nur um den Namen eines Ordners zu ergänzen. Der Befehl Dir System gibt somit das Verzeichnis des System-Ordners der Workbench-Diskette aus.

Da in diesem Fall kein Eintrag mit dem Zusatz Dir versehen ist, handelt es sich bei den angezeigten Namen um echte Dateien. Wir haben also sofort einen letzten Ast erwischt, da keine weiteren Ordner mehr vorhanden sind.

Als zweites Beispiel suchen wir uns einen noch unbekannten Namen aus dem Workbench-Verzeichnis. Der Ordner Devs besitzt kein zusätzliches .info-File, weswegen er auch nicht in Form einer Schublade auf der Workbench dargestellt wird. Vom CLI aus ist es aber nun kein Problem, auch den Inhalt dieses Unterverzeichnisses anzusehen. Der Befehl Dir devs fördert die folgende Liste zutage.

```
keymaps (dir)
printers (dir)
clipboards (dir)
clipboard.device                MountList
narrator.device                 parallel.device
printer.device                  serial.device
system-configuration
```

Man erkennt sofort, daß hier noch drei weitere Äste des Dateibaumes vorhanden sind. Auch der Inhalt eines dieser Unterverzeichnisse kann leicht ausgegeben werden. Die bisherige Pfadangabe (also nur das Devs) muß hierzu nur um einen

Schrägstrich und den Namen des gewünschten Ordners ergänzt werden:

```
dir devs/printers
```

Man darf den Schrägstrich nicht mit dem sog. Backslash (\) verwechseln, der sich unterhalb der F10-Taste befindet. Das Ergebnis dieses Befehls sieht etwa folgendermaßen aus:

```
Alphacom_Alphapro_101          Brother_HR-15XL
CBM_MPS1000                    Diablo_630
Diablo_Advantage_D25           Diablo_C-150
Epson                          Epson_JX-80
generic                        HP_LaserJet
HP_LaserJet_PLUS               ImagewriterII
Okidata_292                    Okidata_92
Okimate_20                     Qume_LetterPro_20
```

Aus diesem Verzeichnis holt sich das Programm Preferences die Daten zur Ansteuerung der verschiedenen Druckertypen. Nun sind keine weiteren Unterverzeichnisse mehr vorhanden. Auf der Workbench-Diskette stellt dies auch die tiefste Schachtelungstiefe für Ordner dar. Noch weiterführende Verzeichnisse könnten jedoch durch Verlängerung des Pfadnamens nach dem gleichen Prinzip erreicht werden (*Dir devs/printers/matrix*).

Zu einer vollständigen Pfadangabe gehört normalerweise auch der Name der Diskette oder die Bezeichnung des Laufwerkes. Im Grundzustand wird aber automatisch immer das Laufwerk Null angesprochen, so daß auf diese Angabe verzichtet werden kann. Wenn Sie zwei oder mehr Laufwerke besitzen, können Sie diese jetzt schon mit dem Dir-Befehl ansprechen. Hierzu muß nur die Laufwerksbezeichnung bündig vor die Pfadangabe gestellt werden. Im einfachsten Fall (keine Pfadangabe) genügt also z.B. *Dir df1:*, um das Hauptverzeichnis einer Diskette auszugeben, die sich im zweiten Laufwerk befindet (Das erste Laufwerk trägt die Bezeichnung df0). Stolze Besitzer einer Festplatte sprechen dieses schöne Gerät mit *Dir dh0:* an. Angaben über Unterverzeichnisse werden immer hinter den Doppelpunkt gehängt:

```
Dir dh0:Texte/Briefe/Rechnungen
```

Wenn Sie nur ein Laufwerk an Ihren Amiga angeschlossen haben, können Sie leider nicht einfach eine beliebige Diskette einlegen und deren Inhaltsverzeichnis ansehen. Das CLI kann ja in so einem Fall den Dir-Befehl nicht mehr finden und verlangt daher in einem Requester sofort nach der Workbench-Diskette. Im dritten Kapitel werden wir uns mit diesem Problem näher auseinandersetzen. Für die Einführung genügt es aber, wenn Sie die Funktionen nur anhand der einen Diskette ausprobieren.

Wo stehen denn nun die CLI-Befehle? Diese Frage ist sehr leicht zu beantworten: im Verzeichnis C der Workbench-Diskette. Der Befehl *Dir C* läßt eine umfangreiche Liste der CLI-Kommandos auf dem Bildschirm erscheinen. Einige Befehle, die sich in ihrem internen Aufbau von den typischen CLI-Befehlen unterscheiden, aber dennoch häufig dazugezählt werden, haben Sie schon oben im System-Ordner gesehen.

1.7 Die Help-Funktion

Eine typische Eigenschaft eines jedes CLI-Befehls ist die eingebaute Hilfsfunktion. Auch einem CLI-Profi kann es immer mal passieren, daß er für einen Befehl die Syntax-Regeln nicht mehr im Kopf hat. Natürlich könnte er nun unser Buch zu Rate ziehen und sich dort in der Quick-Reference über die richtige Anwendung des Befehls informieren. Oft genügt aber auch schon die durch den Befehl selbst zur Verfügung gestellte Format-Zeile. Diese wird ausgegeben, wenn man einen CLI-Befehl mit einem Leerzeichen, gefolgt von einem Fragezeichen eingibt. Beispiel:

```
dir ?
Dir,OPT/K:
```

Bei der Auswertung der durch einen Befehl zur Verfügung gestellten Informationszeile gibt es einige Dinge zu beachten:

Ein Komma trennt mehrere Befehlseinheiten voneinander. Es darf aber später in der eigentlichen Eingabezeile nicht erscheinen. Dir besitzt also zwei solcher Befehlseinheiten: Dir und OPT/K. Die Befehlseinheiten können auch noch bestimmte

Steuerzeichen beinhalten. Diese werden durch den Schrägstrich / eingeleitet. Die zweite Befehlseinheit des Dir-Befehl ist zum Beispiel durch ein K gekennzeichnet. Insgesamt gibt es drei mögliche Steuerzeichen, die folgendermaßen interpretiert werden müssen:

A (Argument)

Für diese Befehlseinheit muß eine Eingabe gemacht werden. Wird sie weggelassen, kann der Befehl nicht ausgeführt werden.

K (Key, Schlüsselwort)

Der Name der Befehlseinheit (im obigen Beispiel: Opt) muß bei der Eingabe erscheinen. Außerdem ist noch ein bestimmter Parameter notwendig. Welche Parameter hier erlaubt sind und welche Funktion hierdurch ausgelöst wird, hängt vom jeweiligen CLI-Befehl ab. Für den Dir-Befehl wird dies ausführlich im zweiten Kapitel besprochen.

S (Switch, Schalterfunktion)

Dieses Steuerzeichen besagt, daß der Name der zugehörigen Befehlseinheit genügt, um eine bestimmte Funktion zu veranlassen. Es darf also kein weiterer Parameter angegeben werden.

Steht in einer Befehlseinheit keines der drei Steuerzeichen, so kann der zugehörige Parameter - falls er angegeben wird - allein durch seine Position innerhalb der Eingabezeile identifiziert werden:

```
dir df0:c
```

Es kann auch vorkommen, daß eine Befehlseinheit gar keinen Namen besitzt. Der Delete-Befehl, auf den an dieser Stelle aber nicht näher eingegangen wird, gibt zum Beispiel die folgende Format-Zeile aus:

```
,,,,,,,,,ALL/S,Q=QUIET/S:
```

Die neun Kommata zu Beginn besagen, daß maximal neun Eingabeparameter ausgewertet werden können. Es besteht aber

keine Eingabepflicht, da keine Steuerzeichen von den Kommata eingeschlossen werden. In dieser Zeile haben wir auch gleich noch eine Besonderheit vorliegen: Einige CLI-Befehle lassen für die Eingabe des Namens einer Befehlseinheit mehrere Möglichkeiten zu. Sie werden in der Format-Zeile durch ein Gleichheitszeichen verbunden (Q = Quiet).

Eine Befehlseinheit, die vom Benutzer durch ihren Namen eingeleitet wird, kann an einer beliebigen Stelle innerhalb der Eingabezeile plaziert werden. Der Copy-Befehl besitzt zum Beispiel unter anderem die Befehlseinheiten From und To/A. Die beiden folgenden Befehlsaufrufe sind daher völlig äquivalent:

```
copy FROM df0:briefe TO df1:texte
```

bzw.

```
copy TO df1:texte FROM df0:briefe
```

Wie im CLI üblich, spielt auch hier die Groß- bzw. Kleinschreibung der Befehlsnamen keine Rolle.

Sehr nützlich ist bei der Hilfsfunktion die Tatsache, daß nach der Ausgabe der Befehlsparameter nicht wieder das DOS-Prompt (1>) erscheint, sondern der Cursor hinter dem Doppelpunkt stehenbleibt und nun auf Eingabe der Parameter wartet. Es ist also nicht notwendig, das gesamte Kommando ein zweites Mal einzugeben, was besonders bei längeren Befehlsnamen sehr lästig wäre. Wenn man sich bei der Arbeit im CLI nicht 100% sicher ist, welche Parameter ein Befehl erwartet, sollte man daher in jedem Fall erst einmal nur das Fragezeichen eingeben.

1.8 Beenden der Arbeit im CLI

Die Funktion des Schließsymbols, mit dem in der Workbench normalerweise Fenster zum Verschwinden gebracht werden, übernimmt im CLI der Befehl EndCLI. Um also die Arbeit im CLI zu beenden, müssen Sie nur diesen Befehl eingeben. Sofort verschwindet das CLI-Fenster.

Durch diese Einführung in die Arbeit mit dem CLI sind Sie mit der grundsätzlichen Bedienung vertraut gemacht worden. In den nun folgenden Kapiteln werden alle derzeit vorhandenen Befehle systematisch durchgesprochen.

2. Die CLI-Befehle und ihre Aufgaben

In diesem Kapitel stellen wir nacheinander jeden CLI-Befehl vor. Wir gehen dabei jedoch nicht alphabetisch vor, sondern nach einer Ordnung, die den Schwierigkeitsgrad und auch die Wichtigkeit des Befehls berücksichtigt. Dadurch soll verhindert werden, daß Sie sofort mit den relativ schwierigen Befehlen konfrontiert werden, die Ihnen den Spaß am Weiterlesen nehmen könnten, da noch die nötigen Hintergrundinformationen fehlen. Trotzdem wird auf eine Unterteilung der Befehle in ihre verschiedenen Aufgabengebiete nicht verzichtet.

Der erste Abschnitt behandelt alle Befehle, die sich unter dem Oberbegriff Laufwerks- und Dateiverwaltung zusammenfassen lassen. Hier finden Sie also die Befehle, die sich auf die Floppy- bzw. Festplattenlaufwerke und die auf ihnen gespeicherten Dateien beziehen.

Im zweiten Abschnitt werden die Befehle besprochen, die sich in irgendeiner Weise auf das System beziehen. Ein typischer Vertreter dieser Gruppe ist das Kommando *Date*, mit dem die Systemzeit aktualisiert werden kann.

Innerhalb des dritten Abschnittes werden speziell die Befehle behandelt, die hauptsächlich in den sog. Batch-Dateien Verwendung finden. Teilweise dürfen sie sogar nur in diesem Zusammenhang auftreten.

Im vierten und letzten Abschnitt schließlich werden zwei recht umfangreiche Befehle besprochen. Die Anführungsstriche stehen, weil man hier eigentlich mehr von Programmen sprechen müßte. Sie heißen ED bzw. Edit und stellen zwei praktische Texteditoren dar. Da ihre Einsatzgebiete recht umfangreich sind, ist ihnen ein eigener Abschnitt gewidmet worden.

Da sich einige CLI-Befehle auf der neuen Workbench-Diskette (Version 1.3) geändert haben, werden wir bei diesen Befehlen zusätzlich noch auf die Änderungen eingehen. Die völlig neu

hinzugekommenen Befehle werden in dem Kapitel 4 ausführlich behandelt.

Die nun folgenden Ausführungen sind keinesfalls als Trockenkurs gedacht. Wir empfehlen Ihnen daher, alle Funktionen, soweit dies möglich ist, direkt im CLI Ihres Rechners nachzuvollziehen. Um einheitliche Voraussetzungen zu schaffen, sollten Sie hierfür die in Kapitel 1 hergestellte, vollständige Kopie der Workbench-Diskette benutzen.

2.1 Befehle zur Laufwerks- und Dateiverwaltung

Zunächst beschäftigen wir uns mit den Befehlen, die für die Organisation und Handhabung der Laufwerke (Disketten, Festplatten, RAM-Disk) und der auf ihnen befindlichen Dateien erforderlich sind. Dabei werden Sie feststellen, daß einige Befehle Funktionen ausführen, die in ähnlicher Form auch von der Workbench aus realisiert werden können. Für viele CLI-Befehle liefert die Workbench jedoch keine vergleichbaren Funktionen.

2.1.1 Format

Syntax: `format DRIVE <disk> NAME <name> [NOICONS]`

Bevor mit einer neuen Diskette oder einer Harddisk das erste Mal gearbeitet werden kann, muß sie speziell vorbehandelt werden. In der Workbench gibt es hierfür den Menüpunkt Initialize. In der Fachsprache heißt der Vorgang, der durch diesen Menüpunkt ausgelöst wird, formatieren. Die genauen Einzelheiten, die sich hierbei auf der Disk abspielen, sollen uns an dieser Stelle nicht interessieren. Man sollte nur wissen, daß bei diesem Vorgang rechnerspezifische Informationen auf die Disk geschrieben werden, die für die spätere Datenorganisation dringend erforderlich sind.

Unvorbehandelte Disketten erkennt das DOS sofort und kennzeichnet sie in der Workbench durch den Namen BAD. Diesen Namen bekommen auch Disketten zugewiesen, die unter einem

anderen Betriebssystem formatiert worden sind (z.B. von einem Atari ST oder einem IBM PC).

Der Befehl zum Formatieren lautet im CLI sinnvollerweise Format. Wie die Syntax-Zeile schon verrät, müssen hinter dem Befehl noch zusätzlich Angaben über das gewünschte Laufwerk und den neuen Namen der zu formatierenden Disk gemacht werden.

Um zum Beispiel eine Diskette im Laufwerk df0 zu formatieren, müßte man eingeben:

```
format DRIVE df0: NAME Spiele NOICONS
```

Der eigentliche Name (hier: Spiele) darf bis zu 30 Zeichen lang sein. Empfehlenswert ist solch ein langer Name aber nicht, da er in der Workbench schnell andere Namen verdeckt. Sollte der Name auch Leerzeichen enthalten, so muß er vollständig in Anführungszeichen gesetzt werden. Das gilt übrigens für die gesamte Arbeit mit dem CLI: Befehlseinheiten dürfen keine Leerzeichen enthalten oder müssen in Anführungsstriche eingeschlossen werden. Beispiel:

```
format DRIVE df0: NAME "Meine Texte" NOICONS
```

Der Zusatz NOICONS bewirkt, daß der Papierkorb, der in dem Disk-Ordner auf der Workbench normalerweise erscheint, nicht erzeugt wird. Er ist für die Arbeit im CLI völlig überflüssig. Falls Sie auf ihn jedoch bei der Arbeit in der Workbench nicht verzichten möchten, lassen Sie den Zusatz NOICONS einfach weg.

Die beiden Wörter DRIVE bzw. NAME müssen bei jedem Formatierungsvorgang angegeben werden. Es genügt also nicht, einfach nur die Laufwerksbezeichnung und den neuen Namen hinter den Befehl zu schreiben. Bei einer Fehleingabe wird dem Anwender sofort die korrekte Syntax des Befehls mitgeteilt. Falls die Eingabe korrekt war, erscheint bei der Formatierung einer Diskette im CLI-Fenster die Meldung:

```
Insert disk to be initialized in drive df0: and press Return
```

Da der Format-Befehl jetzt vollständig in den Arbeitsspeicher des Amiga geladen worden ist, können auch die Besitzer nur einer Diskettenstation in diesem Augenblick die Workbench-Diskette entfernen und die zu formatierende Diskette einlegen. Bevor Sie aber die Return-Taste betätigen, sollten Sie sich noch einmal bewußtmachen, daß bei dem Vorgang alle eventuell schon auf dieser Diskette vorhandenen Daten unwiederbringlich gelöscht werden. Sollte sich bei diesem Gedanken bei Ihnen ein ungutes Gefühl einstellen, haben Sie jetzt noch Gelegenheit, die gesamte Aktion durch ein gleichzeitiges Drücken von Ctrl und C abzubrechen. Ein anschließendes Return beantwortet das CLI dann mit *** *Break.* Mit dem Dir-Befehl, den Sie schon in der Einführung kurz kennengelernt haben, können Sie sich in aller Ruhe ansehen, welche Daten beinahe gelöscht worden wären.

Haben Sie aber den Mut besessen und Return gedrückt, wird die Disk nun formatiert. Im CLI wird dabei gleichzeitig angezeigt, welcher Zylinder gerade in Arbeit ist. Jeder dieser Zylinder besteht aus zwei sich bezüglich der Diskettenoberfläche gegenüberliegenden ca. 0.5 mm breiten, konzentrischen Spuren. Jede Spur zerfällt wiederum in 11 sog. Sektoren, die je 512 Bytes an Daten aufnehmen können. Da eine Diskette insgesamt 80 Zylinder (also 160 Spuren) besitzt, beträgt somit die gesamte Speicherkapazität:

```
(80 * 2 * 11 * 512)/1024 Byte = 880 KByte
```

Falls Sie eine vollständige Kopie einer Diskette anfertigen möchten, ist ein vorheriges Formatieren der Zieldiskette mittels Format nicht notwendig. Der speziell hierfür vorgesehene Disk-Copy-Befehl kann nämlich auch auf fabrikneue Disketten kopieren.

Der Befehl unter Workbench 1.3

Syntax: `format DRIVE <disk> NAME <name> [NOICONS] [QUICK] [FFS] [NOFFS]`

Neu hinzugekommen sind die drei Optionen QUICK, FFS und NOFFS. Durch den Zusatz QUICK benötigt der Format-Befehl für die Ausführung nur noch wenige Sekunden, da lediglich die Spuren formatiert werden, die den Root-Block und die Boot-

Blöcke enthalten. Diese Option dient jedoch nur dazu, eine bereits formatierte und beschriebene Diskette bzw. Festplatte auf eine schnelle Art und Weise wieder löschen zu können. Eine vollständige Formatierung einer Diskette ohne die QUICK-Option benötigt nämlich immerhin knapp zwei Minuten.

Der Root-Block (er befindet sich bei einer Diskette auf Zylinder 40, Seite 0, Sektor 880) stellt die Wurzel der gesamten Verzeichnisstruktur dar. Es wird also lediglich ein leeres Inhaltsverzeichnis auf die Diskette geschrieben, und die von den alten Files belegten Blöcke werden wieder freigegeben. Die Dateien müssen also gar nicht wirklich gelöscht werden.

Außerdem wird durch die Formatierung der Boot-Blöcke (Zylinder 0, Seite 0, Sektor 0 und 1) unter anderem das sich hierauf befindende Boot-Programm erneuert, wodurch ein eventuell schon vorhandener Autostart wieder aufgehoben wird. Nicht zuletzt dient diese Maßnahme auch dazu, sich eventueller Viren zu entledigen, die sich im Frühstadium in den Boot-Blöcken einer Diskette einnisten.

Die beiden Optionen FFS und NOFFS sind eng miteinander verwandt. Sie dienen dazu, beim Formatieren einer Harddisk das gewünschte File-System für die einzelnen Partitions der Festplatte festzulegen. Durch den Zusatz FFS läuft die angegebene Partition unter Verwendung des neuen, erheblich schnelleren FastFileSystem. Falls Sie jedoch NOFFS angeben, benutzt der Amiga das herkömmliche, langsame File-System.

Falls eine Partition unter dem neuen FastFileSystem laufen soll, so muß dies in der sog. Mount-List eingetragen werden. Diese Mount-List findet man im DEVS-Ordner der Workbench-Diskette. Sie kann zum Beispiel mit dem Ed geladen und modifiziert werden (*ed devs:mountlist*). Jede Partition der Festplatte, die nicht schon von vornherein vorhanden (= autokonfiguriert) ist, besitzt hier eine Eintragung. Bevor mit Hilfe des Mount-Befehls neue FFS-Partitions angelegt werden können, müssen zu jeder Partition-Eintragung der Mount-List die folgenden Zeilen hinzugefügt werden:

```
FileSystem = l:FastFileSystem
GlobVec = -1
DosType = 0x4444F5301
StackSize = 4000
```

Wird solch eine Partition innerhalb der Startup-Sequence das erste Mal "gemountet", so erscheint der Requester "Not A DOS Disk ...". Er darf durch Cancel entfernt werden. Anschließend müssen jedoch die unter dem neuen File-System laufenden Partitions neu formatiert werden:

Falls zu einer Partition lediglich die obige FastFileSystem-Eintragung hinzugefügt worden ist, so genügt es, die jeweilige Partition einzeln neu zu formatieren. Ist jedoch auch die Größe der einzelnen Partitions geändert worden (Low- bzw. HighCyl), so ist eine Formatierung des gesamten Festplatte notwendig. Sinnvollerweise sollte man in jedem Fall den Inhalt der zu formatierenden Partitions vorher auf Disketten retten.

2.1.2 Dir

Syntax: `dir DIR,OPT/K`

Dieser Befehl ist Ihnen schon bei der Einführung begegnet. Dort diente er jedoch nur dazu, die Struktur aufzuzeigen, mit der das AmigaDOS die Dateien auf den Disketten bzw. auf der Harddisk verwaltet. In Wirklichkeit leistet dieser Befehl noch viel mehr.

Die Eintragung DIR in der Syntax-Zeile steht für die genaue Pfadangabe des gewünschten Verzeichnisses. Zur Wiederholung:

Falls nicht das aktuelle Gerät (direkt nach dem Start: df0:) angesprochen werden soll, muß zuerst die entsprechende Bezeichnung des gewünschten Gerätes angegeben werden (z.B. df1:, ram:, dh0:, jh0:). Das DOS erkennt Geräteangaben immer an dem abschließenden Doppelpunkt. Anschließend kann eine beliebig tief geschachtelte Angabe folgen, die den Weg zu dem gewünschten Verzeichnis aufzeigt.

Ein Beispiel: Es soll das Verzeichnis Briefe ausgegeben werden, das im Ordner Texte einer sich in Laufwerk 1 befindlichen Diskette ist. Das Kommando hierzu müßte lauten:

```
dir df1:Texte/Briefe
```

Weitere Unterverzeichnisse können durch jeweils einen Schrägstrich getrennt hinten angehängt werden.

Werden keine Angaben gemacht, wird das aktuelle Inhaltsverzeichnis ausgegeben. Welches dies ist, hängt eng mit dem Befehl CD zusammen, der in 2.1.3 ausführlich behandelt wird.

Bis hierhin kannten Sie die Funktion des Dir-Befehls schon aus dem Kapitel 1. Erweitern Sie nun einmal den Befehl um den Zusatz *opt a*. Sie werden feststellen, daß die Ausgabe nun viel umfangreicher ist. Das a steht nämlich für *all* und bewirkt, daß nicht nur das Hauptverzeichnis, sondern alle Unterverzeichnisse, Unter-Unterverzeichnisse usw. ausgegeben werden, bis hin zu den äußersten Zweigen des Dateibaumes. Sehr hilfreich ist diese Option, falls man einmal vor lauter Ordnung eine bestimmte Datei nicht wiederfinden kann. Auf die Festplatte angewandt kann man aber im Datenstrom, der den Bildschirm in kürzester Zeit überflutet, sehr leicht die Orientierung verlieren. Nur ein reaktionsschneller Druck auf eine beliebige Taste gebietet dem Befehl Einhalt. Bekanntlich setzt ein Löschen des Zeichens mit Backspace die Ausgabe fort.

Der Zusatz *opt i* bewirkt eine sogenannte interaktive Ausgabe des Inhaltsverzeichnisses. Diese hochtrabende Bezeichnung besagt nur, daß der Anwender mit in das Geschehen integriert wird. Beim Dir-Befehl sieht das so aus:

Nach jedem angezeigten Datei- oder Ordnernamen hält die Ausgabe an, und ein Fragezeichen erscheint. Sie als Anwender haben nun die Möglichkeit, verschiedene Aktionen über die Tastatur einzuleiten. Durch ein einfaches Drücken der Return-Taste wird die Ausgabe fortgesetzt (weiterhin interaktiv). Durch die Eingabe von Del (Abk. für Delete) kann eine angezeigte Datei von der Diskette gelöscht werden. Das Löschen von Ordnern ist nicht möglich, solange sich noch Dateien oder Unter-

verzeichnisse darin befinden. Sollten Sie es trotzdem versuchen, erscheint im CLI-Fenster die Meldung "directory not empty" und es wird der nächste Eintrag ausgegeben.

Handelt es sich bei dem angezeigten Namen um einen Ordner (Zusatz (Dir) vorhanden!), kann man durch die Eingabe von E (Abk. von Enter) in diesen Ordner gelangen. Mit den Einträgen dieses Ordners wird dann die Ausgabe fortgesetzt. Jederzeit kann nun durch B (Abk. für Back) wieder auf die übergeordnete Verzeichnisebene zurückgekehrt werden. Existiert solch eine Ebene nicht mehr, wird der Dir-Befehl beendet.

Verbirgt sich hinter der angezeigten Datei ein Textfile, kann ihr Inhalt durch die Eingabe von T (Abk. für Type) auf dem CLI-Fenster ausgegeben werden. Sinnvolle Ausgaben erhält man aber wirklich nur bei reinen ASCII-Dateien. Sollte man z.B. auf Programme oder CLI-Befehle die T-Option anwenden, hat dies wirre Ausgaben auf dem Bildschirm zur Folge. Durch gleichzeitiges Drücken von Ctrl und C kann die Ausgabe aber jederzeit unterbrochen werden. Sollte sich die Ausgabe dadurch immer noch nicht normalisieren lassen, hilft Ihnen das Kommando Ctrl und O bestimmt weiter. Hierdurch wird wieder der normale Zeichensatz des Amiga aktiviert.

Die Eingabe eines Fragezeichens läßt alle Antwortmöglichkeiten erscheinen, die in der augenblicklichen Situation vom CLI akzeptiert werden. Gibt man dennoch ein falsches Kommando ein, bekommt man die Mitteilung "Invalid response - try again:?", und man kann es noch einmal probieren.

Der interaktive Dir-Befehl kann jederzeit durch Q (für Quit) beendet werden.

Als letzte Besonderheit gibt es für den Dir-Befehl noch den Zusatz opt ai. Er stellt eine Verknüpfung von opt a mit opt i dar. Es werden also automatisch alle Verzeichnisse und Unterverzeichnisse interaktiv ausgegeben.

Der Befehl unter Workbench 1.3

Syntax: `dir DIR,OPT/K,ALL/S,DIRS/S,INTER/S`

Der neue Befehl bietet im Gegensatz zu dem oben behandelten Dir-Befehl noch die Möglichkeit, die schon besprochenen Optionen A bzw. I auch durch die Angabe der Schlüsselworte *all* bzw. *inter* auszuwählen. Die Eingabe von *dir inter* bewirkt also dasselbe wie die Eingabe von *dir opt i*.

Außerdem kann durch den Zusatz DIRS erreicht werden, daß nur die Namen der Verzeichnisse ausgegeben werden. Auf die Workbench-Diskette angewandt, ergibt der Befehl zum Beispiel die folgende Ausgabe:

```
Trashcan (dir)
C (dir)
System (dir)
inter (dir)
L (dir)
devs (dir)
S (dir)
T (dir)
fonts (dir)
libs (dir)
Empty (dir)
Utilities (dir)
```

Es gibt im interaktiven Modus (*dir opt i* oder *dir inter*) den neuen Befehl *com*. Hierdurch ist es möglich, zwischendurch einen beliebigen CLI-Befehl direkt oder als neuen Prozeß (mit Run) zu starten. Die Ausgabe der Dir-Funktion bleibt solange an der derzeitigen Position stehen.

Sehr nützlich ist diese neue Funktion, falls man zum Beispiel bei seinen Recherchen im Directory auf eine Datei gestoßen ist, die man gerne ausgedruckt haben möchte. Geben Sie dazu einfach den Befehl *com* ein. Der Dir-Befehl fragt Sie anschließend: "Command ?". Nun können Sie den Druckvorgang durch die folgende Eingabe als eigenständigen Hintergrundprozeß starten:

```
run type df0:s/startup-sequence to prt:
```

Anschließend wird der angegebene Text ausgedruckt (in diesem Fall die sog. Startup-Sequence, die sich im Ordner s der Work-

bench-Diskette befindet), während Sie sich gleichzeitig das Inhaltsverzeichnis weiter ansehen können.

2.1.3 CD

Syntax: `cd OrdnerName`

Der Befehl CD (engl.: Change Directory = Ordner wechseln) ermöglicht es, einen bestimmten Ordner einer Diskette oder Festplatte zum aktuellen Ordner des CLI zu machen. Bisher war der aktuelle Ordner das Hauptinhaltsverzeichnis. Daran ändert sich auch nichts, wenn man für einen bestimmten Befehl - wie es in 2.1.1 bei Dir mehrfach vorgekommen ist - einen vollständigen Pfadnamen angibt. Um bei dem Bild des Dateibaumes zu bleiben: Bis jetzt haben wir uns immer vom Boden aus die Äste des Baumes angesehen. Mit CD ist es aber möglich, den Beobachtungsstandort auf einen bestimmten Ast zu verlegen. Danach können bei Pfadangaben alle Beschreibungen entfallen, die genau auf diesen Ast führen. Dazu ein Beispiel:

Wenn Sie nur den Befehl CD eingeben, teilt Ihnen der Rechner mit, auf welchem Ast (bzw. Stamm) Sie derzeit "sitzen". Direkt nach dem Start wird dies im Normalfall der Name der Diskette sein, von der aus das System gestartet worden ist (also z.B. A500 WB1.2 D). Geben Sie nun ein: *CD System.* Ein anschließendes *CD* zeigt nun das neue aktuelle Verzeichnis, nämlich:

```
A500 WB1.2 D:System
```

Alle CLI-Befehle beziehen sich nun auf dieses Unterverzeichnis. Wenn Sie zum Beispiel Dir eingeben, erscheint nicht mehr das Hauptverzeichnis, sondern der Inhalt des System-Ordners. Es ist wichtig, sich den Unterschied zwischen den folgenden beiden Möglichkeiten, diesen Ordnerinhalt auszugeben, ganz klar vor Augen zu halten:

```
1. Möglichkeit:   dir df0:system (Return)
2. Möglichkeit:   cd df0:system (Return)
                  dir (Return)
```

Nach Eingabe der zweiten Variante befindet man sich nicht mehr auf der Ebene des Hauptverzeichnisses, sondern innerhalb des System-Ordners. Das Hauptverzeichnis könnte jedoch nun wieder mit Hilfe der ersten Möglichkeit (*dir df0:*) ausgegeben werden.

Selbstverständlich gibt es aber auch Befehle, die ein Zurückkehren in das Hauptverzeichnis bewirken. Es ist der Befehl CD in Verbindung mit einem oder bei Bedarf mehreren Schrägstrichen. *CD /* läßt uns wieder zum Hauptverzeichnis zurückkehren. Befanden wir uns schon im Unterverzeichnis eines Unterverzeichnisses, müssen wir *CD //* eingeben, um den gleichen Effekt zu erzielen. Wenn man ein sehr tief geschachteltes Unterverzeichnis angewählt hat, kann das Nachzählen der notwendigen Schrägstriche schon ein wenig lästig werden. In einem solchen Fall kann man entweder durch *CD :* oder je nach angeschlossenem Gerät z.B. durch *CD df0:* schnell wieder in das Hauptverzeichnis wechseln. Beide Befehle unterscheiden sich jedoch durch einen kleinen, aber feinen Unterschied, zu dessen Erklärung erst ein bißchen ausgeholt werden muß:

Das AmigaDOS sucht im allgemeinen nicht nur in einem bestimmten Laufwerk nach einer Diskette, sondern behandelt alle Laufwerke gleich. Wenn Sie zum Beispiel irgendeinen Pfadnamen (inklusive des Diskettennamens) eingeben, ist es dem System gleichgültig, in welchem Laufwerk sich solch eine Diskette befindet. Nur in irgendeinem muß sie sich schon befinden. Ist dies nicht der Fall, erscheint sofort ein Requester mit der Aufforderung, doch diese Diskette bitteschön in irgendein Laufwerk einzulegen.

Oft ist es jedoch notwendig oder bequemer, dem System mitzuteilen, welches Laufwerk man speziell ansprechen möchte. Soll zum Beispiel das Hauptverzeichnis der in Drive 0 eingelegten Diskette A500 WB1.2 D zum aktuellen Verzeichnis gemacht werden, könnte man eingeben: *CD "A500 WB1.2 D:"*. Der Amiga würde in diesem Fall auch die Diskette finden, und zwar eben im Laufwerk df0:.

Da es aber unbequem ist, immer den vollständigen Disketten-namen eingeben zu müssen, genügt auch die Eingabe von: *CD df0:*. In diesem Fall muß sich die gewünschte Diskette aber wirklich in Drive 0 befinden, da ja ausdrücklich Laufwerk Null selektiert wurde. Das DOS übernimmt nämlich nun den Namen und identifiziert bei späteren Operationen diese Diskette alleine anhand ihres Namens, so daß sie wieder in ein beliebiges Laufwerk gesteckt werden kann. Hierin liegt nun der grundlegende Unterschied zu dem Befehl *CD :*, denn während *CD df0:* unbedingt auf das Hauptverzeichnis der in Drive 0 eingelegten Diskette zurückspringt, orientiert sich *CD :* immer noch am derzeit aktuellen Ordner. Höchste Zeit für ein Beispiel:

In Laufwerk 0 befindet sich die Workbench-Diskette, in Laufwerk 1 die Diskette Arbeitsdaten mit einem Ordner Kunden. Jetzt gibt man ein: *CD df1:Kunden*. Ein anschließendes *CD* zeigt dann: "Arbeitsdaten:Kunden". Nun können die Disketten in den Laufwerk ohne weiteres vertauscht werden. Dem Amiga ist das gleich. Nun wird der Unterschied zwischen *CD:* und *CD df0:* sehr deutlich: Ein *CD:* macht in jedem Fall das Hauptverzeichnis Arbeitsdaten zum aktuellen Directory, während es bei *CD df0:* davon abhängt, welche der beiden Disketten sich gerade in Laufwerk Null befindet.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.4 MakeDir

Syntax: `makedir /A`

Dieser Befehl übernimmt im CLI in etwa die Aufgabe der Empty-Schublade aus der Workbench. Dort fertigt man ein Duplikat der Schublade an, benennt es beliebig um und hat anschließend die Möglichkeit, in dieser Schublade zusammengehörige Dateien unterzubringen. Ein Beispiel hierfür, das jeder kennt, ist der Ordner DEMOS auf der Workbench-Diskette. Je größer die Kapazität eines Massenspeichers ist, desto mehr ge-

winnt dieser Befehl an Bedeutung. Kam man bei den ca. 170 KByte fassenden C64-Floppys noch mit einem einzigen Verzeichnis aus, ist daran bei den 880 KByte "starken" Amiga-Floppys gar nicht mehr zu denken. Wie wichtig Unterverzeichnisse dann erst für Festplatten mit einer Kapazität von 20, 40 oder noch mehr Megabyte sind, bedarf wohl keiner Erläuterung mehr.

Die Benutzung des MakeDir-Kommandos ist sehr einfach. Es wird nur eine schon vorhandene Pfadangabe benötigt, die um einen Schrägstrich und einen beliebigen neuen Namen erweitert wird:

```
makedir df1:system/monitore
```

Wichtig dabei ist, daß alle Pfade bis auf den zu erstellenden Ordner (hier: monitore) tatsächlich schon existieren. Man kann also nicht den Dateibaum gleichzeitig um einen neuen Ast erweitern, der wiederum schon einen zweiten Ast enthält.

Bei einer Ausgabe des System-Ordners mit Hilfe der Dir-Funktion (*dir df0:system*) wird man anschließend unter anderem auch das neue Verzeichnis monitore wiederfinden. Die Tatsache, daß es sich hierbei tatsächlich um ein Verzeichnis und nicht etwa um eine Datei handelt, macht DOS durch den Zusatz (dir) kenntlich:

```
...
...
monitore (dir)
...
...
```

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.5 Delete

Syntax: `delete ,,,,,,,,,ALL/S,Q=QUIET/S`

Der Befehl Delete ermöglicht es, sich seiner überflüssigen Dateien oder Ordner eines Laufwerkes (dazu zählt auch die RAM-Disk) zu entledigen. Hierfür wird eine vollständige Pfadangabe benötigt, die auf den entsprechenden Namen des zu löschenden Objektes zeigt:

```
delete df0:c/Version
```

Nach Eingabe dieser Zeile würde die Datei Version im Unterverzeichnis c der Diskette im Laufwerk 0 gelöscht, und die durch sie belegten Blöcke würden wieder freigegeben.

Handelt es sich bei Version um einen Ordner, wird dieser nur gelöscht, wenn er keine Dateien oder Unterordner enthält. Andernfalls tritt die Fehlermeldung auf "Not Deleted - directory not empty". In einem solchen Fall - vorausgesetzt, die Daten in dem Ordner sind wirklich überflüssig - kann durch den Zusatz *all* erreicht werden, daß auch der Ordnerinhalt verschwindet. Aber Vorsicht! Schauen Sie lieber noch einmal nach, ob Sie wirklich sämtliche Dateien in dem Ordner nicht mehr benötigen.

Sollte es weder eine Datei noch einen Ordner mit dem Namen Version in dem angegebenen Verzeichnis geben, so wird der Delete-Befehl mit der folgenden Fehlermeldung abgebrochen:

```
Could not get information for "Version" - object not found
```

Sehr nützlich ist bei Delete die Verwendung von Jokerzeichen, die im Kapitel 3 ausführlich besprochen werden. Hat man zum Beispiel gerade die endgültige Version eines Programms fertiggestellt, kann man alle seine Testversionen (test1 bis test7) durch *Delete df0:test#?* auf einmal loswerden (sieben auf einen Streich). Daß auch hierbei äußerste Vorsicht angebracht ist, versteht sich hoffentlich von selbst. Ferner wird bei der Verwendung der Jokerzeichen keine Fehlermeldung ausgegeben, falls es keine Eintragung auf der Diskette gibt, auf die die Beschreibung des Delete-Befehls paßt.

Außer der Benutzung von Jokerzeichen gibt es noch eine weitere Möglichkeit, mehrere Files gleichzeitig zu löschen. Delete wertet nämlich maximal 9 Pfadangaben (s. Syntax-Zeile) aus, die jeweils durch ein Leerzeichen voneinander getrennt hinter dem Befehl stehen können. Beispiel:

```
delete df0:utilities/notepad df1:system/say.info
```

Wie soll man 9 Pfadangaben in eine einzige Zeile quetschen? Gar nicht notwendig! Der Cursor macht sich (bei voller CLI-Fensterbreite) drei Zeilen tiefer bemerkbar, wenn er keine Lust mehr hat, weitere Zeichen anzunehmen. In diesem Fall muß die Tastatur durch mindestens ein Backspace wieder entriegelt werden, da in dem Augenblick auch die Return-Taste ignoriert wird.

Als letzte Besonderheit gibt es bei Delete noch den Zusatz Q bzw. QUIET, der es ermöglicht, die Bestätigungsmeldung für einen Löschvorgang zu unterbinden. Diese tritt immer dann auf, wenn sich das DOS selbst ausrechnen muß, welche Files denn nun eigentlich gelöscht werden müssen. Das ist der Fall bei Verwendung der Jokerzeichen oder des Zusatzes *all*.

Der Befehl unter Workbench 1.3

Die Syntax des Befehls hat sich nicht verändert.

Im Gegensatz zu der alten Version des Delete-Befehls bricht der neue Befehl jedoch beim Löschen mehrerer Eintragungen nicht mehr ab, falls eine der Dateien bzw. Ordner nicht gefunden wird. Gibt man zum Beispiel den Befehl

```
delete df0:test1 df0:test2 df0:test3
```

ein, so wird die Datei test3 auch dann gelöscht, wenn es eine Datei mit dem Namen test2 auf der Diskette nicht gibt. Der alte Delete-Befehl hätte lediglich die Datei test1 gelöscht und anschließend eine Fehlermeldung ausgegeben.

2.1.6 Copy

Syntax: `copy FROM,TO/A,ALL/S,QUIET/S`

Der Befehl Copy ist mit der wichtigste und flexibelste Befehl in der Dateiverwaltung durch das CLI. Er ermöglicht das Kopieren einzelner Dateien oder sogar ganzer Ordner auf ein beliebiges Gerät, das Daten entgegennehmen kann. Natürlich können auch Kopien innerhalb eines Laufwerkes angefertigt werden.

FROM ist eine Pfadangabe zur Quelldatei oder zum Quellordner. Da in der Syntaxzeile hinter FROM kein Zusatz /A vorhanden ist, besteht keine Eingabepflicht. Fehlt die FROM-Angabe, wird automatisch das aktuelle Verzeichnis als Quellordner angesehen.

TO gibt den Zielpfad für den Kopiervorgang an. Hierfür besteht natürlich eine Eingabepflicht. Sollte der Speicherplatz des angegebenen Zielgerätes nicht mehr ausreichen, wird der Kopiervorgang vorzeitig abgebrochen. Eventuelle Dateibruchstücke, die schon kopiert worden sind, werden wieder entfernt. Die zugehörige Meldung lautet:

```
Error while writing
Destination file "xxx" removed
(xxx steht hier für den angegebenen Zielpfad)
```

Welche Angaben für den Zielpfad erlaubt sind, hängt maßgeblich von den Quelldaten ab. Dabei sind die folgenden Situationen möglich:

FROM zeigt auf eine einzelne Datei

In diesem Fall kann der Zielpfad auf ein beliebiges Verzeichnis eines Gerätes oder auch auf ein Gerät selbst zeigen. Handelt es sich bei dem Zielgerät um ein Laufwerk, so wird die Datei im angegebenen Verzeichnis unter dem gleichen Namen weitergeführt. Hierzu zwei Beispiele:

```
copy df0:c/type ram:c
```

Hier wird die Datei type unter dem gleichen Namen in dem Verzeichnis c der RAM-Disk abgelegt. Das c-Verzeichnis muß dazu vorher schon existieren. (Wie man solch einen Ordner erzeugen kann, haben wir bei dem Befehl MakeDir schon besprochen.) Sollte sich schon eine Datei type im c-Verzeichnis der RAM-Disk befinden, wird diese ohne Vorwarnung mit der neuen Datei überschrieben. Dies gilt allgemein bei Namenskollisionen im AmigaDOS. Eine Fehlermeldung "file exists", wie sie manch einer vielleicht noch von anderen (Commodore-) Rechnern kennt, gibt es beim Amiga nicht!

Soll die Kopie jedoch einen anderen Namen als ihr Original bekommen, muß der neue Name zusätzlich noch an den Ordner- bzw. Device-Namen der Zielangabe gehängt werden. Auch hierbei gilt wieder: Eine gleichnamige Datei in diesem Verzeichnis ist verloren. Also Vorsicht!

Existiert schon ein gleichnamiges Unterverzeichnis in dem angegebenen Ziel-Verzeichnis, wird die Kopie in diesem Verzeichnis unter Beibehaltung des alten Namens eingetragen, da ja prinzipiell bei der Eingabe kein Unterschied zwischen Ordner- und Dateinamen gemacht wird.

Auch hierzu wieder ein Beispiel:

```
copy df0:c/Makedir ram:md
```

Es wird der MakeDir-Befehl aus dem C-Verzeichnis unter dem Namen md auf die RAM-Disk kopiert. Der Eintrag md darf jedoch noch nicht als Unterverzeichnis von RAM: existieren. Existiert solch ein Unterverzeichnis, wird der MakeDir-Befehl unter seinem ursprünglichen Namen dort eingetragen.

Wie oben schon erwähnt wurde, ermöglicht der Copy-Befehl die Ausgabe einer Datei auf ein beliebiges Gerät, das Daten empfangen kann. So gibt zum Beispiel der Befehl

```
copy df0:s/startup-sequence prt:
```

die Textdatei Startup-Sequence auf dem Drucker aus.

Ebenso kann durch die folgende Eingabe sehr leicht eine Art Schreibmaschine simuliert werden:

```
copy * to prt:
```

Anschließend kann vom CLI aus zeilenweise ein Text eingegeben werden. Nach jedem Druck auf die Return-Taste wird eine eingegebene Zeile auf dem Drucker ausgegeben. Durch die gleichzeitige Eingabe der Tastenkombination Ctrl und \ wird diese Funktion beendet.

Kommen wir zur zweiten Möglichkeit für die FROM-Angabe.

FROM zeigt auf einen ganzen Ordner

In diesem Fall muß der Zielpfad auf ein vorhandenes Verzeichnis zeigen, da ja mehrere Files kopiert werden sollen. Sollte das DOS das Verzeichnis nicht finden, wird die Fehlermeldung "Output directory not found" ausgegeben.

Leider erlaubt der Copy-Befehl es nicht, beim Kopieren ganzer Ordner den Drucker als Zielgerät anzugeben. Der Copy-Befehl eignet sich also nicht dazu, mehrere Dateien nacheinander auf dem Drucker auszugeben. Bei einem solchen Versuch erscheint die Fehlermeldung:

```
Destination must be a directory
```

Bei der Behandlung des Join-Befehls werden Sie aber auch für dieses Problem eine geeignete Lösung kennenlernen.

Im Normalfall werden nur die echten Dateien des Ordners kopiert. Unterverzeichnisse werden komplett übersprungen. Sollen jedoch auch diese mitkopiert werden, muß bei den Angaben noch der Zusatz ALL erscheinen. Dadurch werden automatisch auch die zusätzlichen Unterverzeichnisse angelegt. Beispiel:

```
copy df0: df1:Spiele all
```

Hierdurch wird eine vollständige File-Copy der Diskette in Laufwerk 0 auf der Diskette in Laufwerk 1 unter dem Ordner-

namen Spiele erzeugt. Das Verzeichnis Spiele muß hierzu vorher schon existieren.

Für die Herstellung von 100%-Kopien von Floppy zu Floppy ist der DiskCopy-Befehl, der im folgenden Abschnitt besprochen wird, in den meisten Fällen die bessere Lösung, da hierbei auch gleichzeitig die Zieldiskette neu formatiert wird. Ab und zu bringt aber die Anfertigung einer solchen Kopie mittels Copy wieder ein wenig Ordnung auf die Diskette, da die auf den Tracks verstreuten, aber logisch zueinander gehörenden Blocks auf der Zieldiskette wieder näher beieinanderliegen und der Lesekopf der Floppy bei einem späteren Zugriff nicht ganz so viele Bewegungen machen muß.

Den Abschluß bildet die QUIET-Funktion, mit der sämtliche Meldungen des DOS über den Kopiervorgang unterdrückt werden können. Diese Option dürfte Ihnen noch von dem Delete-Befehl bekannt vorkommen.

Der Befehl unter Workbench 1.3

Syntax: `copy FROM,TO/A,ALL/S,QUIET/S,BUF=BUFFER/K,`
`CLONE/S,DATE/S,NOPRO/S,COM/S`

Falls beim Kopieren eines Ordners der Zielpfad auf ein nicht vorhandenes Verzeichnis zeigt, so bricht der neue Copy-Befehl nicht mehr ab, sondern er erzeugt selbständig ein gleichnamiges Verzeichnis. In dieses Verzeichnis wird anschließend der gesamte Ordnerinhalt kopiert.

Der neue Copy-Befehl erlaubt auch die Ausgabe eines gesamten Verzeichnisses auf den Drucker. Sinnvollerweise sollten sich in dem Ordner dazu nur echte ASCII-Dateien befinden, da es sonst mit großer Wahrscheinlichkeit zu wirren Ausgaben kommen wird.

Kommen wir nun zu den zusätzlichen Optionen, die in der Befehlszeile des Copy-Befehls auftreten dürfen:

Durch den Zusatz BUFFER bzw. BUF kann die Größe des beim Kopiervorgang benutzten Pufferspeichers in Einheiten zu je 512 Byte vom Anwender festgelegt werden.

Die drei Angaben CLONE, DATE, NOPRO und COM bestimmen, welche zusätzlichen Informationen des ursprünglichen Files oder Ordners mit in die Kopie übernommen werden sollen. Diese zusätzlichen Informationen, die das DOS für alle Dateien und Ordner bereitstellt, sind Angaben über den Zeitpunkt des letzten Schreibzugriffs und bestimmte Statusinformationen, deren Aufgabe bei der Behandlung des Protect-Befehls näher erläutert wird. Bei Dateien kann ferner noch ein bis zu 80 Zeichen langer Kommentar zu diesen Informationen gehören.

Der Befehl, mit dessen Hilfe man sich diese Angaben ansehen kann, heißt List. Er wird im nächsten Abschnitt ausführlich besprochen.

Für den Copy-Befehl müssen Sie nun wissen, daß durch den Zusatz CLONE alle Angaben, durch DATE das Datum und durch COM der Kommentar (falls vorhanden) mit in die Kopie übernommen wird. Die Option NOPRO funktioniert genau umgekehrt: Sie verhindert, daß die alten Statusinformationen übernommen werden.

Möchten Sie zum Beispiel eine Datei Test nur unter Beibehaltung des Datums und des Kommentars in die RAM-Disk kopieren, würde dies der folgende Befehl für Sie erledigen:

```
copy Test ram: DATE COM NOPRO
```

2.1.7 List

Syntax: `list DIR,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K,SINCE/K, UPTO/K,QUICK/S`

Vielleicht waren Sie zu Beginn enttäuscht, wie wenig Informationen man mittels des Dir-Befehls über die Dateien und die Floppy bekommen kann. Schließlich teilt einem schon der betagte C64 mit, wieviel Platz auf der Diskette noch frei ist oder

wie viele Blöcke ein File belegt. Das kann der Amiga natürlich auch. Geben Sie hierzu nur den Befehl List gefolgt von dem gewünschten Pfadnamen ein, beispielsweise:

```
list df0:
```

Das Ergebnis wird Sie sicher zufriedenstellen. Bei der Workbench-Diskette sieht es folgendermaßen aus:

```
Directory "df0:" on Sunday 15-May-88
Expansion.info              346 rwed 02-Mar-87 23:29:10
Trashcan                    Dir rwed 02-Mar-87 23:29:10
.info                        82 rwed 09-Jun-87 12:43:46
c                           Dir rwed 02-Apr-87 09:44:21
Clock.info                  338 rwed 02-Mar-87 23:31:49
Demos                       Dir rwed 15-Apr-87 07:57:18
Clock                     19668 rwed 02-Mar-87 23:32:20
System                      Dir rwed 06-Apr-87 15:11:06
l                           Dir rwed 02-Mar-87 23:33:26
devs                        Dir rwed 02-Mar-87 23:35:49
s                           Dir rwed 16-Apr-87 08:43:34
t                           Dir rwed 06-Apr-87 15:08:45
Preferences.info            418 rwed 02-Mar-87 23:36:00
Preferences               58136 rwed 02-Mar-87 23:36:20
Demos.info                  346 rwed 02-Mar-87 23:36:22
fonts                       Dir rwed 02-Mar-87 23:37:32
libs                        Dir rwed 02-Mar-87 23:37:59
Empty                       Dir rwed 09-Jun-87 12:46:57
Utilities.info              346 rwed 02-Mar-87 23:38:08
Disk.info                   306 rwed 02-Apr-87 09:05:36
System.info                 346 rwed 02-Mar-87 23:38:13
Empty.info                  346 rwed 02-Mar-87 23:38:16
Trashcan.info               430 rwed 02-Mar-87 23:38:17
Utilities                   Dir rwed 15-Apr-87 07:57:45
Expansion                   Dir rwed 15-Apr-87 07:35:10
12 files - 13 directorys - 196 blocks used
```

Gehen wir die Liste der Reihe nach durch: Zuerst einmal haben wir mitgeteilt bekommen, welches Inhaltsverzeichnis überhaupt gezeigt wird und daß man mal wieder dabei ist, einen seiner freien Sonntage vor dem Rechner zu verbringen. Die Datumsangabe kann natürlich nur stimmen, wenn man entweder mit dem Programm Preferences das Datum korrekt eingestellt hat oder einen Amiga mit batteriegepufferter Echtzeituhr besitzt.

Im Gegensatz zum Dir-Befehl folgen nun unsortiert die File- bzw. Ordnernamen, die sich im Hauptverzeichnis der Diskette befinden. Handelt es sich um echte Files, wird auch deren

Größe in Bytes angegeben. Ansonsten steht an dieser Stelle die Abkürzung Dir für Directory.

Datum und Uhrzeit geben den Zeitpunkt des letzten Schreibzugriffs auf den entsprechenden Eintrag wieder.

Bleibt nur noch die Erklärung der seltsamen Buchstabenkombination rwed. Jeder Buchstabe stellt eine bestimmte Statusinformation des zugehörigen Files dar. Dabei bedeuten:

r	(Read)	Soll das Lesen erlauben.
w	(Write)	Soll Schreibzugriffe erlauben.
e	(Execute)	Soll das Ausführen einer Datei erlauben.
d	(Delete)	Der Eintrag kann gelöscht werden.

In den obigen Beispiel-Dateien ist also alles erlaubt. Wenn ein oder mehrere Punkte verboten sind, erscheint an der betreffenden Stelle ein Strich. Eine Datei mit der Kombination rwe- z.B. ist vor unbeabsichtigtem Löschen geschützt.

Die drei anderen Flags (rwe) sind allerdings bei der derzeitigen DOS-Version des Kickstart 1.2 noch nicht implementiert (daher das "soll" bei den Erklärungen). Das DOS kümmert sich überhaupt nicht darum, was hier eingetragen ist, sondern läßt die genannten Aktionen immer zu.

Der Befehl Protect, der später noch behandelt wird, ermöglicht eine Beeinflussung dieser Statusinformationen.

Am Ende der List-Ausgabe erscheinen noch die Anzahl Files bzw. Ordner und der durch sie belegte Speicherplatz in Blocks (2 Blocks = 1 KByte).

Wie die Syntax-Zeile des List-Befehls vermuten läßt, müssen noch einige Optionen des Kommandos erläutert werden. Meistens genügt einem der List-Befehl jedoch ohne diese zusätzlichen Angaben. Trotzdem ein Überblick über die Möglichkeiten dieses Befehls:

Dir

Soll nicht das aktuelle Inhaltsverzeichnis ausgegeben werden, kann man auch einen anderen Pfad vorgeben (z.B. List ram:c). Im Gegensatz zum Dir-Befehl darf dieser Pfad auch auf eine einzelne Datei zeigen.

PAT (engl. Pattern = Muster)

Hier kann vorgegeben werden, nach welchen Einträgen gesucht werden soll. Dazu sind die Jokerzeichen (s. Kapitel 3) sehr nützlich. Beispiel: List pat a#? gibt nur Einträge aus, die mit einem a beginnen.

KEYS

Für normale Anwendungen ist diese Option relativ uninteressant. Es wird zusätzlich noch in eckigen Klammern der Start-Block des jeweiligen Eintrags auf dem Laufwerk ausgegeben.

DATES

Aktiviert die Datumsanzeige. Liegt der Schreibzugriff auf eine Datei oder einen Ordner weniger als eine Woche zurück, so gibt das DOS in solchen Fällen normalerweise anstelle des Datums den Wochentag im Klartext aus. Zugriffe, die nur einen Tag zurückliegen, oder die am selben Tag stattgefunden haben, kennzeichnet das DOS durch den Kommentar *Yesterday* bzw. *Today*. Durch den Zusatz DATES erreicht man nun, daß auch bei diesen Eintragungen das Datum im Format TAG-MONAT-JAHR ausgegeben wird.

NODATES

Die Datums- und Uhrzeitausgabe wird vollständig unterdrückt.

TO

Es kann eine Datei oder ein Device (z.B. prt: für den Drucker) zur Umlenkung der Ausgabe angegeben werden.

S (Subname)

Diese Option trägt nur zur Verwirrung bei. Sie soll es ermöglichen, nach Einträgen zu suchen, die an irgendeiner Stelle im Namen mit dem angegebenen Subnamen übereinstimmem. Ein Subname ist dabei nichts anderes als ein Teil eines Namens. Wie in Kapitel 5 zu lesen ist, läuft die Eingabe von #?subname#? in Verbindung mit der Option PAT (s.o.) auf das gleiche hinaus.

SINCE

Diese praktische Funktion zeigt alle Einträge an, die nach dem angegebenen Zeitpunkt erstellt worden sind. Dadurch kann man z.B. lediglich die Namen der Textdateien der letzten zwei Tage ansehen. Das Datumsformat muß dabei so angegeben werden, wie es bei der Ausgabe durch List erscheint (TT-MMM-JJ). Natürlich möchte der Amiga dabei englisch angesprochen werden. (Also z.B. für März nicht Mär, sondern Mar eingeben!)

UPTO

Analog zu SINCE kann man hierdurch eine Datums-Obergrenze eingeben.

Selbstverständlich können SINCE und UPTO auch kombiniert benutzt werden. Außerdem versteht der Amiga die Begriffe Yesterday und Today sowie die Wochentage der vergangenen Woche. Beipiel:

```
list since monday upto yesterday
```

QUICK

Es werden nur noch die Namen der Einträge ausgegeben. Einziger Vorteil gegenüber dem Dir-Befehl: Die Anzahl der Dateien und die Anzahl der insgesamt belegten Blöcke werden am Ende angezeigt.

Zum Schluß eine kleine Übung: Was stellt der nun folgende Befehl an?

```
list df1:kundenkartei/#?mann nodates to prt: since 10-sep-86 upto
today
```

Lösung: Es werden im Unterverzeichnis Kundenkartei der Diskette im Laufwerk 1 genau die Verzeichnis- und Dateinamen ohne Datumsangabe auf dem Drucker ausgegeben, die mit "mann" enden. Bedingung ist noch, daß sie nicht vor dem 10. September 1986 abgespeichert worden sind.

Der Befehl unter Workbench 1.3

Syntax: `list DIR,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K,SINCE/K,
UPTO/K,QUICK/S,BLOCK/S,NOHEAD/S,FILES/S,DIRS/S,LFORMAT/K`

Es sind einige sehr nützliche Optionen hinzugekommen:

Wird der List-Befehl mit dem Zusatz BLOCK aufgerufen, zeigt er die Größe der Files nicht mehr in Bytes, sondern in Disketten-blöcken an.

Die Option NOHEAD unterdrückt die Ausgabe des Verzeichnisnamens und des aktuellen Datums. Diese Angaben erscheinen immer dann, wenn dem List-Befehl ein Verzeichnisname mitgegeben wird (z.B. list df0:). Außerdem verhindert NOHEAD die Ausgabe der Abschlußmeldung (xx files - yy directories - zz block used).

Benötigt man nur die Namen der Unterverzeichnisse eines Ordners, kann man den List-Befehl durch den Zusatz DIRS ergänzen. Ebenso ist es möglich, sich nur die Namen der Dateien ausgeben zu lassen. Die Option hierfür lautet FILES.

Geradezu genial ist die neue Option LFORMAT. Hiermit ist es möglich, die Ausgabe des List-Befehls so zu formatieren, daß sie zum Beispiel direkt als Anweisungstext für eine Batch-Datei verwendet werden kann. Die Ausgabe der Dateinamen erfolgt dabei wie bei Verwendung der Optionen QUICK und NOHEAD. Hinter der Option LFORMAT wird das Format für die Ausgabe angegeben.

```
list df0: LFORMAT="..."
```

Innerhalb der Anführungszeichen muß angegeben werden, aus welchen Angaben eine List-Zeile bestehen soll. Hier kann ein beliebiger Text stehen. Die Zeichenkombination %s hat dabei jedoch eine besondere Bedeutung:

Taucht die Kombination %s innerhalb der Anführungszeichen einmal auf, so wird an dieser Stelle der File-Name eingefügt. Beispiel: Eingabe:

```
list df0:c LFORMAT="Dies ist der %s-Befehl"
```

Ausgabe:

```
Dies ist der Run-Befehl
Dies ist der Fault-Befehl
Dies ist der Install-Befehl
Dies ist der Stack-Befehl
Dies ist der Prompt-Befehl
Dies ist der Else-Befehl
Dies ist der Status-Befehl
Dies ist der Ed-Befehl
Dies ist der BindDrivers-Befehl
(...)
```

Möchte man zum Beispiel bei allen Dateien des Verzeichnisses mit dem Namen Texte das Delete-Flag löschen, kann dazu mit Hilfe von List die folgende Batch-Datei erstellt werden:

```
list >Batch_Datei df0:Texte LFORMAT="protect %s -d"
```

Die Batch-Datei könnte anschließend folgendermaßen aussehen:

```
protect Text_1 -d
protect Text_2 -d
protect Text_3 -d
protect Brief_1 -d
protect Brief_2 -d
```

Diese Datei kann durch den Befehl *EXECUTE Batch_Datei* direkt ausgeführt werden.

Die Zeichenkombination %s darf innerhalb der Anführungszeichen auch mehrmals auftauchen. Werden zwei %s verwendet, so wird an beiden Stellen jeweils der File-Name eingefügt. Bei drei %s steht anstelle der letzten beiden %s der File-Name und an-

stelle des ersten %s die Pfadangabe zum aktuellen Verzeichnis. Beispiel:

Eingabe:

```
list >Batch_Datei c: LFORMAT="copy %s%s to Ordner/%s.BAK"
```

Ausgabe:

```
copy c:Run to Ordner/Run.BAK
copy c:Fault to Ordner/Fault.BAK
copy c:Install to Ordner/Install.BAK
copy c:Stack to Ordner/Stack.BAK
copy c:Prompt to Ordner/Prompt.BAK
copy c:Else to Ordner/Else.BAK
(...)
```

Diese Batch-Datei erzeugt zu jedem CLI-Befehl eine Kopie in dem Verzeichnis mit dem Namen Ordner.

Werden bei der LFORMAT-Angabe vier %s-Steuerzeichen verwendet, so werden sie nacheinander ersetzt durch die Pfadangabe, den File-Namen, die Pfadangabe und den File-Namen.

Beim List-Befehl haben sich noch andere Änderungen ergeben.

Die Jokerzeichen werden jetzt viel flexibler gehandhabt. Mit dem neuen List-Befehl ist es endlich möglich, schon bei der Pfadangabe die Jokerzeichen zu verwenden. Beispiel:

```
list df0:c/m#?
```

Dieses Kommando würde die Namen sämtlicher Dateien des C-Verzeichnisses ausgeben, die mit einem m beginnen. Der alte List-Befehl würde hier nur lapidar melden:

```
Can't examine "df0:c/m#?": object not found
```

Schließlich hat sich bei der Ausgabe der Status-Informationen noch eine Änderung ergeben. Zusätzlich zu den schon vorhandenen fünf Status-Flags rwed werden durch die Workbench 1.3 noch vier weitere Flags eingeführt. Sie werden gekennzeichnet durch die Buchstaben h (Hidden), s (Script), P (Pure) und A

(Archive). Diese Flags werden beim Protect-Befehl ausführlich besprochen.

2.1.8 Rename

Syntax: `rename FROM/A,TO=AS/A`

Als angehender CLI-Profi können Sie sich sicher schon denken, was man mit diesem Befehl anstellen kann. Der Name sagt eigentlich schon alles: Man kann seinen Dateien einen neuen Namen geben.

Ohne Parameter ist Rename natürlich sinnlos. Es müssen zwei Angaben erscheinen (daher das /A in der Syntaxformel):

1. Vollständige Pfadangabe zum umzubenennenden Objekt.
2. Der neue Pfadname.

Auf den ersten Blick scheint alles ganz einfach zu sein: *rename texte/mein.text texte/Aufsatz* benennt auch tatsächlich die Datei mein.text um in Aufsatz. In Wirklichkeit ist der Rename-Befehl aber viel flexibler. Das obige Beispiel ist nur ein Spezialfall, bei dem der Pfad an sich gleich geblieben ist. Man kann nämlich auch eine Datei oder einen Ordner innerhalb des Dateibaumes vollständig verlagern. Dabei muß wieder einmal unterschieden werden:

1. Das umzubenennende Objekt ist eine einzelne Datei.

In diesem (einfacheren) Fall muß der Zielpfad aus einer Ordnerangabe gefolgt von einem (eventuell) neuen Namen für die Datei bestehen. Beispiel:

```
rename df0:system/format df0:c/formatieren
```

Hier wird der CLI-Befehl Format aus dem System-Ordner in den C-Ordner unter dem Namen Formatieren eingetragen. Da das Unterverzeichnis C schon vorher existiert hat, klappt die Umbenennung tatsächlich. Gibt es keinen C-Ordner oder kann der Rename-Befehl den Format-Befehl nicht finden, erfolgt die Fehlermeldung: "Can't rename system/format as c/formatieren"

2. Das umzubenennende Objekt ist ein Ordner.Will man tatsächlich nur den Ordnernamen umändern, reicht eine Eingabe wie bei der einfachen Dateiumbenennung aus. Beispiel:

```
rename df0:demos df0:Demoprogramme
```

Nun kann man aber auch einen ganzen Ordner samt Inhalt an eine andere Stelle des Dateibaumes bewegen. Auch für diesen Fall haben wir uns ein Beispiel einfallen lassen: Nehmen wir an, auf einer Diskette befindet sich ein Verzeichnis BASIC, das ein Unterverzeichnis Programme enthält. Außerdem existiere im Hauptverzeichnis ein Ordner Spiele mit dem Unterverzeichnis Adventures. Nun könnte man auf die Idee kommen, den kompletten Spiele-Ordner (samt Unterverzeichnis) in den Programm-Ordner einzutragen. Genau das macht der nun folgende Befehl:

```
rename df0:SPIELE to df0:BASIC/PROGRAMME/SPIELE
```

Wie man sieht, muß dabei auch gleichzeitig ein Name für den neuen Ordner angegeben werden. Da Spiele aber immer noch sinnvoll ist, haben wir es dabei belassen. Das to kann hier auch wegfallen. Bei unserem Experimenten mit dem Rename-Befehl in Verbindung mit der RAM-Disk sind wir auf einen Fehler des sog. RAM-Handler gestoßen: Im Normalfall darf es niemals vorkommen, daß innerhalb eines Verzeichnisses ein Name doppelt vorkommt. Der MakeDir-Befehl z.B. weigert sich in einem solchen Fall: xxx already exists (Eintrag xxx existiert schon). Geben Sie nun einmal die folgenden Befehle ein:

```
makedir ram:test
makedir ram:test/unter
rename  ram:test/unter ram:test dir ram:
```

Ein ganz schön dicker Hund: Das Verzeichnis test gibt es nun zweimal im Hauptverzeichnis der RAM-Disk. Zum Glück kann man durch ein einfaches Rename eines der beiden Verzeichnisse schnell wieder umbenennen. Noch ein letzter Hinweis: Es ist mit dem Rename-Befehl nicht möglich, eine Verlagerung einer Datei oder eines Ordners von einem Laufwerk auf ein anderes vorzunehmen. Die folgende Eingabe ist daher verboten und ergibt eine Fehlermeldung:

```
rename df0:c/type ram:type
```

Diese Spezialität bleibt dem Copy-Befehl vorbehalten.

Der Befehl unter Workbench 1.3

An der Funktion des Rename-Befehls sind keine Änderungen vorgenommen worden. Aufgrund einer Verbesserung des RAM-Handlers ist es nun jedoch nicht mehr möglich, in einem Verzeichnis der RAM-Disk zwei Dateien den gleichen Namen zu geben.

2.1.9 DiskCopy

Syntax: `diskcopy [FROM] <disk> TO <disk> [NAME <name>]`

Wohl eine der ersten Maßnahmen eines frischgebackenen Amiga-Besitzers ist die Anfertigung von Sicherheitskopien der im Lieferumfang enthaltenen Disketten. Auf der Workbench geschieht dies durch Anklicken des Icons der entsprechenden Disk, gefolgt von einer Aktivierung des Menüpunktes Duplicate. Im Gegensatz zu dem schon besprochenen CLI-Befehl Copy wird dabei eine vollständig gleichwertige Kopie der Diskette angefertigt. Bis auf sehr wenige Ausnahmen, auf die wir später noch zu sprechen kommen, finden sich alle Informationen der Original-Disk anschließend an den gleichen Stellen auf dem neuen Datenträger wieder. Es ist nicht einmal notwendig, die Zieldiskette vorher mittels Format zu formatieren. Dies geschieht automatisch während des Kopiervorgangs. Sollten sich auf der Zieldiskette schon Daten befunden haben, sind diese unwiederbringlich verloren.

Die Funktion des Duplicate-Befehls für Disketten übernimmt im CLI der Befehl DiskCopy. Bei der Ausführung der Funktion werden Sie aber auf so ein schönes Fenster, wie es bei diesem Vorgang auf der Workbench auftaucht, verzichten müssen.

Um zum Beipiel ein Duplikat einer Diskette unter Verwendung nur eines einzigen Laufwerkes anzufertigen, muß man vom CLI

aus den folgenden Befehl eingeben (to muß stehen, from kann weggelassen werden):

```
diskcopy df0: to df0:
```

Ist man sich sicher, daß die eventuell schon vorhandenen Daten auf der Zieldiskette wirklich nicht mehr benötigt werden, kann man den Vorgang nach dem Einlegen der Quelldiskette durch Drücken der Return-Taste starten. Gibt man jedoch vorher die Tastenkombination Ctrl und C ein, wird beim folgenden Return die Funktion vorzeitig verlassen und die folgende Meldung ausgegeben:

```
*** BREAK
Disk Copy Abandoned.
Remember to insert original disk
Disk Copy Terminated
```

Abandoned besagt, daß der Vorgang vorzeitig abgebrochen worden ist. Da sich bei einem Abbruch der Funktion während des Schreibvorganges ja noch die Zieldiskette im Laufwerk befindet, die nun nichts Halbes und nicht Ganzes enthält, wird man zusätzlich noch daran erinnert, wieder die Original-Diskette einzulegen. Die Meldung "Disk Copy Terminated" erscheint auch nach einer vollständigen Ausführung des Befehls und weist noch einmal darauf hin, daß die Funktion nun beendet ist.

Wie auch auf der Workbench wird einem vor dem Start des Kopiervorgangs in Extremfällen noch mitgeteilt, wie oft die Diskette gewechselt werden muß. Bei einer Ausgabe der Meldung:

```
"The Disk Copy will take 12 swaps."
```

sollte man sich besser nach einem Diskjockey umsehen, der einem diese Arbeit abnimmt. Können Sie jedoch für kurze Zeit auf Ihre Textverarbeitung und das 4096-Farben-Zeichenprogramm verzichten, kommt man hier auf akzeptablere Werte. Im günstigsten Fall erreicht man z.B. mit dem Amiga 500 in der Grundausrüstung (1/2 MByte) eine Kopie mit dreimaligem Diskettenwechsel.

Mit einem zweiten Laufwerk kennt man solche Probleme nicht. Der ernsthafte Anwender wird sich - falls er es nicht schon längst getan hat - sowieso mit der Zeit nach einem weiteren Massenspeicher umsehen, da das dauernde Diskettenwechseln auf die Dauer nervtötend ist (und zwar nicht nur beim DiskCopy-Befehl).

Kommen wir nun auf die Punkte zu sprechen, die eine mit DiskCopy erstellte Kopie von ihrem Original unterscheiden:

In der Syntax-Liste ist der Zusatz [Name <name>] noch gar nicht beachtet worden. Es ist nämlich möglich, der Kopie einen anderen Namen zu geben, der natürlich irgendwo auf ihr abgespeichert wird: Unterschied Nummer 1! Dazu wieder ein Beispiel, diesmal aber für zwei Laufwerke:

```
diskcopy drive df0: to drive df1: name "Workbench 1.2"
```

Aber auch ohne die Angabe eines Namens kann DOS jederzeit die Kopie von dem Original unterscheiden, nämlich anhand des Datums und der Uhrzeit des Kopiervorgangs. Diese Daten werden auf einer Kopie immer aktualisiert abgespeichert: Unterschied Nummer 2!

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.10 ReLabel

Syntax: `relabel DRIVE/A,NAME/A`

Mit Hilfe dieses Kommandos können Sie Ihren Disketten einen neuen Namen geben. Die folgende Zeile demonstriert, wie man eine Diskette, die sich in diesem Fall im Laufwerk 1 befinden muß, in Spiele umbenennen kann:

```
relabel df1: Spiele
```

Das Leerzeichen hinter der Laufwerksbezeichnung darf dabei nicht fehlen. Enthält der eigentliche Name Leerzeichen (wie z.B. der Name der Workbench-Diskette: A500 WB1.2 D), muß man ihn in Anführungszeichen setzen. Die maximal erlaubte Länge eines Diskettennamens beträgt 30 Zeichen. Auf der Workbench ist so ein langer Name aber eher hinderlich.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.11 Info

Dies ist zumindest in der Anwendung einer der einfachsten Befehle, denn er benötigt keine weiteren Parameter. Er könnte auch genausogut im Kapitel 2.2 zu finden sein, da er Systemfunktions-Charakter besitzt. Da sich die durch ihn in Erfahrung zu bringenden Informationen jedoch auf die Laufwerke beziehen, haben wir ihn in dieses Kapitel aufgenommen.

Nach Eingabe von Info erhält man einige Angaben über die angeschlossenen Laufwerke. Solch eine Ausgabe sieht beispielsweise folgendermaßen aus:

```
Mounted disks:
Unit Size    Used    Free Full Errs  Status     Name
DF0: 880K    1645     113  93%   0   Read Only  A500 WB1.2 D
DF1: 880K     534    1224  30%   0   Read/Write Textomat
RAM:   2K       4       0 100%   0   Read/Write RAM DISK
Volumes available:
A500 WB1.2 D [Mounted]
RAM DISK [Mounted]
Texte
Textomat [Mounted]
```

Im ersten Teil erhält man Informationen über alle angeschlossenen (mounted) Laufwerke, die dem System bekannt sind. In diesem Fall sind es zwei Floppy-Laufwerke und die RAM-Disk. Die Kapazität einer Diskette (Size: 880 KByte) haben wir bei dem Format-Befehl schon einmal berechnet.

Unter Used und Free findet man die Anzahl Blocks (2 Blocks = 1 KByte), die jeweils belegt bzw. noch frei sind. Falls man damit nichts anfangen kann, erhält man zusätzlich noch den Belegungsgrad in Prozent. Da die RAM-Disk ihren Speicher dynamisch verwaltet und dadurch immer nur soviel RAM vom Systemspeicher abzweigt, wie sie für ihren derzeitigen Inhalt benötigt, erhält man bei ihr immer einen Belegungsgrad von 100%. Es ist also egal, ob sich in der RAM-Disk zum Beispiel ein 2 KByte großes Textfile oder aber eine 150 KByte umfassende Kundendatei befindet: Die RAM-Disk ist in beiden Fällen zu 100% gefüllt und kann scheinbar keine weiteren Daten mehr aufnehmen (Free: 0). In Wirklichkeit bestimmt natürlich der noch verfügbare Arbeitsspeicher über die Kapazität der RAM-Disk. Lediglich die Angaben unter Size bzw. Used sind für der Anwender wirklich informativ.

Defekte Blocks (Errors) gibt es auf den Beispieldisketten keine. Vor allem für Besitzer einer Festplatte ist diese Angabe sehr interessant, da sie hieran ungefähr erkennen können, wieviel Zeit ihnen noch bleibt, für ein neues Laufwerk zu sparen.

Status gibt den Zustand des Schreibschutzclips der zugehörigen Diskette wieder. Auf die Diskette in Laufwerk 0 kann also nur lesend zugegriffen werden. Zuletzt wird noch der Name ausgegeben, unter dem man derzeit auch auf das jeweilige Laufwerk zugreifen kann. Diese Namen ändern sich natürlich, sobald man die Disketten wechselt. Lediglich der Name der RAM-Disk kann sich bei der Workbench Version 1.2 nicht ändern.

Unter der Überschrift Volumes available werden nun noch die Namen der Disks ausgegeben, die dem System bekannt sind. Dies sind in erster Linie natürlich die Namen der sich in den Laufwerken befindenden Disketten. Sie werden mit dem Zusatz [Mounted] versehen. Außerdem tauchen in dieser Liste die Namen der Disketten auf, die zwar derzeit nicht eingelegt sind und daher nicht angesprochen werden können, aber auf die von irgendeinem Programm oder auch vom DOS aus noch ein Zugriff besteht (zum Beispiel über die Befehle Assign oder Path, die noch besprochen werden).

Der Befehl unter Workbench 1.3

Syntax: `info DEVICE`

Der neue Info-Befehl kann um eine Geräteangabe erweitert werden. Dadurch erhält man lediglich Informationen über das spezifizierte Gerät. Mit der Einführung längerer Gerätenamen (z.B. FFS0:) ist auch eine Überarbeitung der Tabellenausgabe notwendig geworden. Der Info-Befehl unter Workbench 1.2 würde nämlich bei einem Gerät mit einem längeren Namen die folgende unschöne Ausgabe ergeben.

```
Mounted disks:
Unit Size    Used    Free Full Errs Status     Name
FFS0: 30M   33043   28397  46%   0  Read Only  FAST0
DF1: 880K    1714      44  97%   0  Read Only  A500 WB 1.2
DF0: 880K    1649     109  93%   0  Read/Write A500 WB 1.3

Volumes available:
A500 WB 1.2 D [Mounted]
FAST0 [Mounted]
Textomat
A500 WB 1.3 [Mounted]
```

Eine Art Tabulatorfunktion verhindert beim neuen Info-Befehl diesen Effekt. (Der neue Lock-Befehl, der in Kapitel 3 besprochen wird, ermöglicht übrigens den Status *Read Only* auch für Festplatten.)

2.1.12 Install

Syntax: `install DRIVE/A`

Bei einem Neustart stellt sich der Amiga ganz schön wählerisch an: Er reagiert im Normalfall nur bei der Workbench-Diskette mit einem Ladevorgang. Man kann ihn mit den teuersten Disketten füttern, die große Amiga-Hand will einfach nicht verschwinden. Natürlich hat das normalerweise nichts mit der Qualität der Disketten zu tun. Es liegt vielmehr einfach daran, daß sie nicht durch den Install-Befehl zu selbstladenden (=bootbaren) Disketten umgewandelt worden sind.

Mit *Install df0:* wird zum Beispiel die in Drive 0 eingelegte Diskette bootbar gemacht. Dazu muß sie jedoch zumindest vorher schon formatiert worden sein. Besitzer einer Harddisk werden enttäuscht darüber sein, daß ein Systemstart von der Festplatte aus (noch) nicht vorgesehen ist. Diese Möglichkeit besteht erst bei Verwendung der neuen Betriebssystemversion Kickstart 1.3 und spezieller Harddisks, die hierfür vorbereitet sind. Da das Betriebssystem beim Amiga 500 bzw. 2000 fest in ein ROM eingebrannt ist, wird man bei der älteren Rechnergeneration wohl auf den Systemstart mittels Harddisk verzichten müssen.

Wird der Befehl auf eine frisch formatierte Diskette angewandt und mit dieser anschließend das System neu gestartet, landet man automatisch - und zwar relativ schnell - im CLI. Das mit der Schnelligkeit muß natürlich irgendeinen Haken haben. Es hat nicht nur einen, sondern ganz viele:

Das fängt schon mit der Darstellung der Zeichen an. Bei Verwendung eines Monitors möchte man nämlich normalerweise mit 80 und nicht nicht mit nur 60 Zeichen pro Zeile arbeiten. Sicher werden Sie sich noch daran erinnern, daß das Programm Preferences zwei Felder zeigt, mit deren Hilfe man die gewünschte Darstellungsart auswählen und diese Einstellung durch Save auf der eingelegten Diskette abspeichern kann. Diese Daten werden dabei in einem Unterverzeichnis mit dem Namen devs unter der Bezeichnung System-Configuration abgelegt. Sie können sich nun diese Datei mit dem Copy-Befehl von der Workbench auf die leere Diskette in einen durch den Befehl MakeDir erzeugten Ordner devs kopieren. Nach einem Reset steht Ihnen dann der Amiga nach knapp sieben Sekunden im 80-Zeichen-Modus zur Verfügung.

Das war aber nur das geringste Übel: Der Rechner hat auch noch eine falsche Tastaturbelegung, denn anstelle der Umlaute findet man auf den entsprechenden Tasten verschiedene Sonderzeichen. Dieses Problem kann leider nicht durch Veränderungen der Preferences behoben werden. Hier müssen wir schon etwas tiefer in die Trickkiste greifen:

Außer der System-Konfiguration befindet sich in dem devs-Verzeichnis der Workbench-Diskette unter anderem noch ein Ordner mit dem Namen keymaps. Jede Datei in diesem Unterverzeichnis enthält Informationen über die speziellen Tastaturbelegungen des Amiga in den verschiedenen Ländern. Den File-Eintrag für die von uns gewünschte Belegung kann man hier unter d finden. Nun genügt es aber nicht, diese Datei in einen entsprechenden Ordner auf unsere Diskette zu kopieren, denn das System wählt von alleine keine solche Tastaturbelegung aus. Die amerikanische Tastaturbelegung ist nämlich als Default- (= Grund-) Einstellung im Kickstart-ROM (bzw. beim Amiga 1000 auf der Kickstart-Disk) vorhanden. Wir benötigen daher noch ein Programm, das eine Tastaturumstellung ermöglicht. Dies befindet sich auf der Workbench-Diskette im System-Ordner und heißt SetMap. Haben Sie auch solch einen Ordner auf der neuen Boot-Diskette angelegt und SetMap hineinkopiert, versuchen Sie noch einmal einen neuen Boot-Vorgang.

Es wird sich nichts verändert haben, da niemand das SetMap-Programm aufgerufen hat. Geben Sie daher vom CLI aus ein:

```
system/setmap d
```

(Das Y finden Sie beim Z, der Schrägstrich liegt auf der Minus-Taste, rechts neben dem Punkt.)

Endlich sind nun auch die Umlaute wieder vorhanden. Aber warum muß man das alles von Hand eingeben? Schließlich ist nach dem Boot-Vorgang mit der Workbench-Diskette ja auch sofort die deutsche Tastaturbelegung vorhanden. Die Ursache hierfür soll nur kurz erwähnt werden:

Bei einer bootbaren Diskette kann mit einem einfachen Textverarbeitungsprogramm (z.B. dem ED aus dem C-Ordner) ein Textfile erzeugt werden, das beliebige CLI-Kommandos enthalten darf, die beim Booten automatisch abgearbeitet werden. Solche Dateien werden allgemein Batch-Files genannt. Ihnen ist in diesem Buch ein eigenes Kapitel gewidmet. Für uns reichen an dieser Stelle die folgenden Informationen:

Das Batch-File muß den Namen Startup-Sequence tragen und sich im Unterverzeichnis s befinden. DOS führt ein solches File dann bei einem Reset automatisch aus. Wenn Sie sich einmal mit dem im nächsten Abschnitt behandelten Type-Befehl die Startup-Sequence der Original Workbench-Diskette ansehen, werden Sie hier auch das setmap d wiederfinden.

Für eine sinnvolle Mindestkonfiguration fehlen unserer Diskette jetzt nur noch ein paar CLI-Befehle. Diese kann man sich nach Belieben von einer anderen Diskette holen. Wichtig ist nur, daß man sie in einen Ordner mit dem Namen c ablegt, da das DOS nur hier selbständig nach ihnen sucht.

Der Befehl unter Workbench 1.3

Syntax: `install DRIVE/A,NOBOOT/S,CHECK/S`

Neu hinzugekommen sind die beiden Optionen NOBOOT und CHECK. Durch NOBOOT kann die Installation einer Diskette nun auch wieder rückgängig gemacht werden. Höchste Zeit wurde auch die Einführung der Option CHECK, mit der man testen kann, ob eine bestimmte Diskette bootbar ist oder ob der Boot-Block Unregelmäßigkeiten aufweist. Diese Unregelmäßigkeiten könnten zum Beispiel durch die Infizierung mit einem Computer-Virus verursacht werden. Diese mehr oder weniger gefährlichen Viren werden von ihren Entwicklern meist so programmiert, daß sie sich zumindest in den Boot-Block einer Diskette einnisten und sich anschließend selbst in die Boot-Blöcke aller im späteren Verlauf der Arbeit neu eingelegte Disketten kopieren. Werden so behandelte Disketten weitergegeben, kann sich der Virus im Schneeballsystem weiterverbreiten.

Bei einer nicht bootbaren Diskette können solche Viren nichts anstellen, da sie niemals die Kontrolle über den Rechner bekommen. Der CHECK-Befehl gibt bei nicht bootbaren Disketten die Meldung aus:

```
No bootblock installed
```

Handelt es sich bei der eingelegten Diskette um eine Boot-Disk, deren Boot-Block keine Unregelmäßigkeiten aufweist, lautet die Meldung:

```
Appears to be normal V1.2/V1.3 bootblock
```

Gibt der Install-Befehl dagegen die folgende Meldung aus:

```
May not be standard V1.2/V1.3 bootblock
```

dann ist irgend etwas faul im Staate Dänemark. Sollte es sich um eine gewöhnliche, von Ihnen formatierte Diskette handeln, hat sich Ihr Rechner mit großer Wahrscheinlichkeit einen Virus eingefangen. Die Folgen solcher Viren sind recht unterschiedlich und reichen von einem reinen Selbstzweck (der Virus stellt gar nichts an) über den Besuch kleiner Männchen auf dem Bildschirm bis hin zu der gefürchteten Guru Meditation samt Formatierung der Festplatte. Genauso verschieden wie die Symptome sind auch die Möglichkeiten der Therapie.

Eine Möglichkeit, die vielleicht Erfolg verspricht, wollen wir kurz beschreiben: Schalten Sie zunächst den Rechner mittels des Netzschalters für einen Augenblick ab (mind. 5 Sekunden), und booten Sie ihn anschließend mit einer Diskette, von der Sie mit hundertprozentiger Sicherheit wissen, daß sie nicht durch einen Virus verseucht ist. Da wohl jeder Anwender bei der ersten Arbeit eine Sicherheitskopie der neuen Workbench-1.3-Diskette angefertigt hat, wird diese Diskette in den meisten Fällen die genannte Bedingung erfüllen. Starten Sie also ausnahmsweise den Amiga mit dieser Diskette, und öffnen Sie ein CLI-Fenster. Geben Sie nun die folgenden Befehle ein:

```
dir >nil: ram:
copy c:install ram:
path ram: add
```

Bringen Sie nun die Workbench-Diskette an einen sicheren Ort zurück. Sie können nun, auch mit nur einem Laufwerk, alle Ihre Disketten mittels des Kommandos *install df0: check* auf das Vorhandensein eines Virus testen und den Boot-Block gegebenenfalls durch *install df0:* neu installieren. Wenn Sie diesen Vorgang mit allen Disketten durchgeführt haben, sollten Sie

nach einigen Diskettenoperationen noch einmal die Boot-Blöcke kontrollieren. Leider erwischt man nämlich durch diese Prozedur nur die dummen Viren, die sich ausschließlich in den Boot-Blöcken niederlassen. Wirklich ausgeklügelte Viren nisten sich zusätzlich noch in anderen Teilen der Diskette ein (z.B. in das trackdisk.device) und sind daher so gut wie gar nicht auszumachen. In einem solchen Fall sollten Sie sich möglichst schnell an einen Computer-Club wenden, der Ihnen bestimmt bei Ihrem Problem weiterhelfen kann.

2.1.13 Type

Syntax: `type FROM/A,TO,OPT/K`

Bei der interaktiven Ausgabe eines Diskettenverzeichnisses mit *Dir opt i* haben wir schon die Möglichkeit kennengelernt, reine Textdateien auf dem Bildschirm ausgeben zu lassen. Genau hierzu dient auch der Befehl Type. Um zum Beispiel den Text mountlist im Unterverzeichnis devs der Workbench-Diskette anzusehen, bedarf es der folgenden Eingabe:

```
type df0:devs/mountlist
```

Aber aufgepaßt: Die Ausgabe hält nicht von alleine an, wenn der Text nicht auf eine Bildschirmseite paßt. In einem solchen Fall genügt jedoch die Betätigung irgendeiner Taste, um den Vorgang zu stoppen. Für die Fortsetzung der Ausgabe empfehlen wir die Backspace-Taste (Pfeil nach links), da hierbei anschließend wieder ein einfaches DOS-Prompt (1) erscheint.

Sie möchten das Ergebnis auf Papier haben? Eine unserer leichtesten Übungen, denn als einen weiteren Parameter kann für eine Umlenkung der Ausgabe noch ein vollständiger Pfadname (TO) angegeben werden. Erweitert man die Eingabe um den Zusatz prt:, geht die Ausgabe auf den Drucker. Selbstverständlich kann auch eine Datei als Ausgabemedium dienen:

```
type devs:mountlist t/Meintext
```

Nun enthält die Datei Meintext genau das, was sonst auf dem Bildschirm erschienen wäre. Daß so etwas funktioniert, dürfte niemanden mehr vom Stuhl reißen, denn eine Umlenkung der Ausgaben eines CLI-Befehls ist mittels des Zeichens > jederzeit möglich. Die folgende Zeile ist daher zur obigen äquivalent:

```
type >t/Meintext devs:mountlist
```

Sollen die Zeilen bei der Ausgabe automatisch durchnumeriert werden, kann man dies durch das Anhängen von opt n erreichen. So kann zum Beispiel ein BASIC-Programm, das im ASCII-Modus abgespeichert worden ist, noch nachträglich mit Zeilennummern versehen werden, um anderen Personen eine spätere Eingabe zu erleichtern.

Hauptsächlich für Profis ist opt h (für hexadezimal) vorgesehen. Der Type-Befehl führt nämlich nur dann zu einer einwandfreien Textausgabe, wenn die Datei reine ASCII-Zeichen und keine Steuerzeichen enthält. Ein DOS-Kommando zum Beispiel besteht als echtes Programm vorwiegend aus Zeichen, die zu wirren Ausgaben führen: Der Befehl *type c/type* erzeugt nichts als Hektik auf dem Bildschirm. Der obige Befehl, ergänzt durch ein opt h, zaubert folgende Tabelle ins CLI-Fenster (Ausschnitt):

```
0000: 000003F3 00000000 00000002 00000000    ................
0010: 00000001 0000004F 000001C4 000003E9    .......O........
0020: 0000004F 286A0164 700C4E95 2401223C    ...O(j.dp.N.$."<
0030: 00000095 49FAFFEE 286CFFFC 2F0C2F02    ....I...(l.././.
```

Ganz rechts haben wir unseren Text. Man erkennt, daß für den einfachen Type-Befehl wirklich nicht viel Sinnvolles dabei ist. Jedes Pünktchen steht für ein nicht darstellbares Zeichen, es sei denn, es handelt sich ausnahmsweise wirklich einmal um einen Punkt.

In der ersten Spalte werden die Zeichen hexadezimal durchnumeriert. In der Mitte steht nun der Inhalt des Files, dargestellt durch jeweils vier Langworte. Ein solches Langwort wiederum besteht aus vier Bytes, jedes Byte stellt genau ein Zeichen dar, so daß jeweils genau ein Byte mit einem rechts dargestellten Zeichen korrespondiert.

Das I in der letzten Zeile zum Beispiel befindet sich an der 53. Byte-Position (= 3*16 + 4). Der ASCII-Code, der zur Verschlüsselung von Texten benutzt wird, lautet für ein "I": $49 ("$" = hexadezimal) oder dezimal 73 (4*16+9).

Eine Kombination der Optionen N und H ergibt wenig Sinn und ist daher verboten. Probiert man es doch, erscheint die Fehlermeldung: Invalid option combination N & H.

Der Befehl unter Workbench 1.3

Syntax: `type FROM/A,TO,OPT/K,HEX/S,NUMBER/S`

Ein Unterschied gegenüber dem alten Type-Befehl besteht darin, daß die Optionen opt h und opt n nun auch durch den alleinigen Zusatz HEX bzw. NUMBER aktiviert werden können. Besipiel:

```
type s:startup-sequence number
```

anstelle von

```
type s:startup-sequence opt n
```

2.1.14 Join

Syntax: `join ,,,,,,,,,,,,,,,AS/A/K`

Steht man vor dem Problem, mehrere Dateien zu einer einzigen zusammenfassen zu müssen, bietet der Join-Befehl die richtige Lösung hierfür. Mit ihm lassen sich maximal 15 beliebige Dateien gleichzeitig miteinander verknüpfen.

Die 15 Kommata in der Syntax-Zeile stehen für die maximal mögliche Anzahl an Quelldateien. Das Schlüsselwort AS muß angegeben werden (/K). Anschließend wird eine Pfadangabe für die Ergebnis-Datei erwartet (/A). Im einfachsten Fall kann mit Join die Grundfunktion des Type-Befehls simuliert werden. Hierzu gibt man nur eine einzige Quelldatei an und lenkt die

Ausgabe mittels * auf den Bildschirm. Das folgende Beispiel gibt den Text der Startup-Sequence auf dem Bildschirm aus:

```
join df0:s/startup-sequence as *
```

Es stört nicht, wenn das gleiche File mehrmals unter den Quelldateien zu finden ist. Dabei fällt uns sofort der Drucker ein: Bisher haben wir noch kein Kommando kennengelernt, das es uns erlaubt, für einen Druckvorgang mehrere Dateien gleichzeitig angeben zu können. Der Copy-Befehl akzeptiert zwar die Jokerzeichen, man kann aber nicht gezielt mehrere Dateien durch ihren wirklichen Namen ansprechen. Der Join-Befehl macht es jetzt aber möglich, bis zu 15 Dateien nacheinander ausdrucken zu lassen. Beispiel:

```
join text1 text3 text 4 text5 as prt:
```

Ergebnis: Es werden die Textdateien text1 bis text5 nacheinander auf dem Drucker ausgegeben.

Auch für Programmierer von Compiler-Sprachen bietet sich manchmal der Join-Befehl an: Sollte es ein Editor nicht mehr schaffen, alle Programmteile gleichzeitig in seinen Arbeitspeicher aufzunehmen, können diese auch jeweils einzeln bearbeitet und erst direkt vor dem Compilieren zu einer einzigen Datei zusammengelinkt werden.

Der Befehl unter Workbench 1.3

Syntax: `join ,,,,,,,,,,,,,,,AS=TO/K`

Der Join-Befehl versteht jetzt auch To anstelle von As.

2.1.15 Search

Syntax: `search FROM,SEARCH/A,ALL/S`

Der Befehl Search ermöglicht es, beliebige Dateien auf das Vorhandensein einer vorgegebenen Zeichenkette zu untersuchen. Wird eine passende Zeichenkette gefunden, erscheint der Name

der entsprechenden Datei, gefolgt von der Zeilennummer und der Zeile, die den Such-String enthält.

FROM steht hier wieder für einen vollständigen Pfadnamen auf ein Verzeichnis oder eine einzelne Datei. Fehlt diese Angabe, geht der Befehl von dem aktuellen Verzeichnis aus. In diesem Fall muß vor dem Such-String noch das Wort SEARCH eingefügt werden, damit der String nicht mit einem Pfadnamen verwechselt wird. Beispiel:

```
search search "Commodore AMIGA"
```

Hierdurch werden alle Dateien des aktuellen Verzeichnisses auf den Text Commodore AMIGA durchsucht. Es wird dabei nicht zwischen Groß- und Kleinschrift unterschieden. Die Anführungszeichen müssen immer dann stehen, wenn der Such-String Leerzeichen enthält. Sollen auch alle Unterverzeichnisse durchsucht werden, kann dies durch den Zusatz all erreicht werden. Der Befehl zeichnet sich durch ein paar Besonderheiten aus:

Wie beim neuen List-Befehl besteht auch hier die Möglichkeit, den Pfadnamen durch Jokerzeichen zu ergänzen. Der Befehl

```
search ???/#?t hallo
```

würde zum Beispiel alle Dateien, die mit t enden und sich in einem Unterverzeichnis befinden, dessen Name aus drei beliebigen Buchstaben besteht, auf das Vorkommen des Wortes hallo untersuchen.

Wie alle CLI-Befehle kann auch Search durch die Tastenkombination Ctrl und C abgebrochen werden. Beim Suchvorgang in ganzen Verzeichnissen besteht ferner noch die Möglichkeit, durch gleichzeitiges Drücken von Ctrl und D sofort mit der nächsten Datei fortzufahren.

Die bei manchen Suchvorgängen auftretende Meldung: "Line x truncated" besagt, daß die gerade bearbeitete Datei aus zu langen Zeilen besteht.

Eine nützliche Hilfe ist der Search-Befehl auch für die Programmierer der Sprache C. Sie können hiermit z.B. schnell herausfinden, in welchem Include-Verzeichnis sich eine gesuchte Datenstruktur befindet.

Der Befehl unter Workbench 1.3

Syntax: `search FROM,SEARCH/A,ALL/S,NONUM/S,QUICK/S`

Bei dem neuen Search-Befehl wurde die Meldung "Line xx truncated" sinnvollerweise umgewandelt in "Warning: line xx too long". Weiterhin wird jetzt der Fehlercode 5 zurückgeliefert, falls der Suchvorgang negativ verlaufen ist (sonst: Null). Den Fehlercode kann man innerhalb der sog. Batch-Dateien, denen in diesem Buch ein eigenes Kapitel gewidmet ist, für bestimmte Zwecke auswerten.

Schließlich sind noch zwei neue Optionen hinzugekommen:

NONUM

Unterdrückt die Ausgabe der Zeilennummer, falls die Search-Funktion fündig geworden ist. Da außerdem der gefundene Text linksbündig ausgegeben wird, kann man bei umfangreicheren Suchvorgängen die Fundstellen besser erkennen.

QUICK

Durch diesen Zusatz werden bei der Ausgabe lediglich die Namen der durchsuchten Files nebeneinander anstatt untereinander ausgegeben. Erst wenn die Funktion auf das nächste Verzeichnis stößt, wird eine neue Zeile begonnen.

2.1.16 Sort

Syntax: `sort FROM/A,TO/A,COLSTART/K`

Relativ wenig Beachtung findet im allgemeinen dieser leistungsfähige Befehl. Sort ermöglicht das alphabetische Sortieren einer reinen Textdatei.

Die Parameter FROM und TO erklären sich fast von selbst:

FROM

Gibt den Pfadnamen zu der zu sortierenden Datei an. Da diesmal keine Verzeichnisse zugelassen sind, ist eine Eingabe zwingend erforderlich (/A).

TO

Muß entweder ein Pfadname zu einer Datei oder der Name eines Devices sein, das die sortierten Daten entgegennehmen kann. Die FROM-Datei wird nämlich nicht verändert.

Möchte man die Ausgabe z.B. auf den Bildschirm bekommen, muß hier das Sternchen (*) angegeben werden. Durch prt: werden die Daten auf den Drucker umgeleitet.

COLSTART

Bei Bedarf kann durch diesen Zusatz die Spalte (engl.: column) festgelegt werden, ab der in den einzelnen Zeilen mit dem Sortiervorgang begonnen werden soll. Hat man es zum Beispiel mit einem Text zu tun, der in jeder Zeile die ersten 10 Stellen für den Vornamen und eine beliebige Anzahl Stellen für den Nachnamen reserviert, kann durch den Zusatz Colstart 11 eine Sortierung nach den Nachnamen erreicht werden. Fehlt eine COLSTART-Angabe, so wird ab der ersten Spalte sortiert.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.17 Protect

Syntax: `protect FILE/A,FLAGS:`

Können Sie sich noch an die merkwürdige Buchstabenkombination rwed erinnern? Falls nicht, dann geben Sie doch noch einmal den List-Befehl ein. Dämmert es wieder? Es handelt sich

dabei um das schon in Kapitel 2.1.7 angesprochene Statuswort, das die erlaubten Zugriffsarten auf eine Datei oder einen Ordner angibt.

Kurze Wiederholung:

r	steht für Read;	soll das Lesen erlauben.
w	steht für Write;	soll einen Schreibzugriff erlauben.
e	steht für Execute;	soll ein Execute erlauben.
d	steht für Delete;	ermöglicht das Löschen des Eintrages.

Mit dem Protect-Befehl ist es nun möglich, das Statuswort neu einzugeben. Derzeit wird aber nur das Delete-Flag tatsächlich vom DOS beachtet. Dieses Flag stellt sozusagen den softwaremäßigen Schreibschutzclip für jeden einzelnen Eintrag auf der Diskette dar. Alle anderen Flags werden vollkommen ignoriert. So kann zum Beispiel eine Datei auch dann ausgelesen werden, wenn das Read-Flag nicht gesetzt ist. Analoges gilt für das Write- und das Execute-Flag.

Wenn Sie zum Beispiel den Ordner Briefe auf der Diskette im Laufwerk 1 vor unbeabsichtigtem Löschen schützen möchten, können Sie dies folgendermaßen erreichen:

```
protect df1:Briefe
```

Da keine Status-Flags angegeben wurden, werden alle Flags zurückgesetzt. Viel Sinn ergibt das jedoch nicht, da ein Ordner, der mindestens eine weitere Datei oder einen Unterordner enthält, ohne weiteres sowieso nicht gelöscht werden kann (Fehlermeldung: directory not empty) und gegen ein Löschen des Ordnerinhaltes mit Hilfe des Befehls *delete briefe all* auch dieser Schreibschutz nicht ankommt. Es kann lediglich verhindert werden, daß der leere Ordner aus Versehen gelöscht wird. Die einzelnen Dateien innerhalb eines Verzeichnisses können jedoch durch ihr eigenes Statuswort geschützt werden:

```
protect df1:Briefe/Einladung
```

Nun kann die Datei Einladung nicht mehr gelöscht werden. Das zugehörige Statuswort besteht jetzt aus vier Minuszeichen (----), die besagen, daß (theoretisch) kein Zugriff mehr auf diese Datei

getätigt werden kann. Einzig und alleine der Versuch, die Datei zu löschen, wird jedoch vom DOS mit einer Fehlermeldung (hier: file ist protected from deletion) quittiert. Aufgehoben wird der Schutz durch:

```
protect df1:Briefe/Einladung rwed
```

Nun ist der ursprüngliche Zustand wiederhergestellt, und die Datei kann gelöscht werden.

Der Befehl unter Workbench 1.3

Syntax: `protect FILE/A,FLAGS,ADD/S,SUB/S`

Mit der Workbench 1.3 sind vier neue Flags eingeführt worden:

H (Hidden)
S (Script)
P (Pure)
A (Archived)

Bei Verwendung der Kombination Workbench 1.3/Kickstart 1.2 werden jedoch lediglich die beiden Flags P und S beachtet.

Ein gesetztes Hidden-Flag unterdrückt im Inhaltsverzeichnis die Eintragung des zugehörigen Files. So können zum Beispiel die .info-Dateien, die in erster Linie für das Icon auf der Workbench verantwortlich sind, in der Directory-Liste unsichtbar gemacht werden. Dadurch werden besonders längere Directorys wesentlich übersichtlicher.

Das Script-Flag kennzeichnet Batch-Files. Ist das Script-Flag gesetzt, so kann das Batch-File direkt von einer Shell aus gestartet werden. Es ist also nicht mehr notwendig, den Befehl Execute vor den Namen des Batch-Files zu schreiben. Der Execute-Befehl wird bei gesetztem Script-Flag automatisch aufgerufen (s. Execute-Befehl).

Ein gesetztes Pure-Flag besagt, daß das zugehörige Programm mit Hilfe des Befehls Resident geladen werden kann. Dadurch steht es dem Anwender jederzeit sofort zur Verfügung, muß also nicht mehr von einem Laufwerk nachgeladen werden.

Das Pure-Flag ist notwendig, da noch längst nicht jedes Programm die Eigenschaften mitbringt, die für die Anwendung des Resident-Befehls notwendig sind. Die Programme müssen nämlich sowohl reentrant als auch mehrfach aufrufbar sein. Erfüllt das Programm diese Eigenschaften, so sollte der Programmierer dies durch ein Setzen des Pure-Flags kenntlich machen. Bis auf wenige Ausnahmen sind sämtliche CLI-Befehle der Workbench 1.3 mit dem Pure-Flag versehen. Weitere Informationen zu dem Thema finden Sie bei der Behandlung des Resident-Befehls in Kapitel 4.

Das Archive-Flag erleichtert unter Kickstart 1.3 die Anfertigung von Sicherheitskopien von Dateien. Der Copy-Befehl kopiert aus einem Verzeichnis nur solche Dateien, deren Archive-Flag nicht gesetzt ist. Anschließend wird jedoch das Flag gesetzt und damit die Datei als archiviert gekennzeichnet. Bei einem erneuten Schreibzugriff auf die Datei wird das Archive-Flag automatisch gelöscht. Dadurch wird dem Copy-Befehl bei der späteren Anfertigung von Sicherheitskopien signalisiert, daß eine Veränderung an der Datei stattgefunden hat und eine erneute Archivierung dieser Datei notwendig ist.

Dazu eine Anwendung: Arbeitet man mit der schnellen RAM-Disk, so könnte man vom CLI aus eine Batch-Datei als Hintergrundprozeß aktivieren, die in regelmäßigen Abständen (z.B. alle 5 Minuten) alle bis dahin veränderten Dateien auf eine Diskette rettet. Legt man die dazu notwendigen Befehle Copy, Wait und Execute auch noch resident im Arbeitsspeicher ab, so bemerkt man eine eventuelle Rettungsaktionen lediglich an dem Anlaufen der Diskettenstation. Eine solche Batch-Datei mit dem Namen BAKUP_BATCH könnte folgendermaßen aussehen:

```
wait 5 min
copy ram:#? to df0:
execute BACKUP_BATCH
```

Natürlich funktioniert diese Batch-Datei auch unter Kickstart 1.2, jedoch wird dabei der gesamte Inhalt der RAM-Disk gerettet, unabhängig davon, ob in den letzten 5 Minuten ein Schreibzugriff auf die RAM-Disk stattgefunden hat oder nicht.

Neu hinzugekommen sind beim Protect-Befehl die beiden Optionen ADD bzw. SUB. Damit ist es jetzt auch möglich, einzelne Flags gezielt zu setzen bzw. zurückzusetzen. Die Angabe des volständigen Statuswortes ist also nicht mehr unbedingt notwendig. Durch ADD kann ein einzelnes Flag gesetzt und durch SUB ein Flag zurückgesetzt werden. Beispiel:

```
SUB
Status vorher     ----rwed
Eingabe           protect datei d sub
Status nachher    ----rwe-

ADD
Status vorher     ----rwe-
Eingabe           protect datei d add
Status nachher    ----rwed
```

Weiterhin können die Optionen ADD und SUB auch durch ein dem Status-Flag vorangestelltes Plus- bzw. Minuszeichen ersetzt werden. Hierdurch wird die Eingabe noch einmal vereinfacht:

```
-
Status vorher     ----rwed
Eingabe           protect datei -w
Status nachher    ----r-ed

+
Status vorher     ----r-ed
Eingabe           protect datei +w
Status nachher    ----rwed
```

2.1.18 FileNote

Syntax: `filenote FILE/A,COMMENT/A`

Möchte sich ein Programmierer noch nachträglich in seinem Werk verewigen oder auch nur eine Notitz über die aktuelle Versionsnummer hinterlassen, steht ihm hierzu der FileNote-Befehl zur Verfügung. Dieses Kommando erlaubt es, eine beliebige Datei mit einem bis zu 80 Zeichen langen Kommentar zu versehen. Lesen kann man diesen Kommentar dann mit dem List-Befehl. Er erscheint bei der Ausgabe unter der entsprechenden Datei in einer eigenen Zeile, die zur Unterscheidung

von den übrigen Dateinamen durch einen Doppelpunkt eingeleitet wird. Dazu ein Beispiel:

Man vergißt andauernd, wozu der FileNote-Befehl dient. Eine kleine Notiz soll daher als Gedächtnisstütze dienen. Der zugehörige Befehl sieht fogendermaßen aus:

```
filenote c:filenote "Hiermit kann man einen bis zu 80 Zeichen langen Kommentar zu jeder Datei schreiben!"
```

Die Anführungszeichen müssen immer dann stehen, wenn der Text (wie hier) Leerzeichen enthält. Wird anschließend der List-Befehl auf die Datei c:filenote angewandt, erscheint die folgende Ausgabe:

```
c:filenote                    700 rwed 02-Mar-87 23:30:19
: Hiermit kann man einen bis zu 80 Zeichen langen Kommentar zu
jeder Datei schreiben!
```

Noch eine Anmerkung: Bei einem Kopiervorgang mit Copy wird eine eventuell vorhandene Dateinotiz nicht mitkopiert. Sollte die Zieldatei schon exisitieren, wird die eigentliche Datei zwar durch den Kopiervorgang überschrieben, eine eventuell vorhandene Dateinotiz jedoch nicht. Nur bei Verwendung des Copy-Befehls der Workbench-Version 1.3 kann man frei entscheiden, ob eine eventuell vorhandene Dateinotiz mitkopiert werden soll oder nicht. Nähere Informationen hierzu finden Sie im Abschnitt 2.1.6, in dem der Copy-Befehl ausführlich behandelt wird.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.19 SetDate

Syntax: `setdate FILE/A,DATE,TIME`

Dieser Befehl ist besonders für diejenigen Amiga-Anwender gedacht, die ihren Rechner noch nicht mit einer batteriegepufferten Echtzeituhr bestückt haben und ein tägliches Einstellen des Datums und der Uhrzeit mit dem Programm Preferences zu lä-

stig finden oder diese Prozedur einfach manchmal vergessen. Der Befehl erlaubt es nämlich, die beim Abspeichern von Dateien bzw. Ordnern mit abgelegten Zeitangaben noch nachträglich zu korrigieren.

In der Syntax-Zeile steht FILE für eine vollständige Pfadangabe auf eine Datei oder einen Ordner. Dahinter muß sich das gewünschte Datum befinden, und zwar in dem gleichen Format, wie es beim List-Befehl benutzt wird:

```
setdate texte/Brief 13-may-88
```

Außerdem kann das Datum des gestrigen bzw. heutigen Tages auch durch die Angabe *yesterday* bzw. *today* gesetzt werden. Ferner ist es möglich, durch die direkte Angabe der englischen Bezeichnung des Wochentages das Datum um maximal sechs Tage bezüglich der derzeitigen Systemzeit vorzudatieren. Voraussetzung hierfür ist natürlich, daß die Systemzeit vorher z.B. mit Hilfe des Programms Preferences auf den aktuellen Stand gebracht worden ist.

Zusätzlich zu der aktuellen Datumsangabe sollte auch jedesmal die Uhrzeit mit angegeben werden, da sonst in die Datei automatisch 0 Uhr eingetragen wird. Datumsangaben, die vor dem 2. Januar 1978 liegen, werden übrigens nicht übernommen. In solchen Fällen erscheinen in der Ausgabe zwei leere Felder.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.1.20 DiskDoctor

Syntax: `diskdoctor DRIVE/A`

Zu den unangenehmsten Meldungen des Betriebssystems gehört sicherlich der in den unpassendsten Situationen auftauchende System-Requester, der darauf hinweist, daß mit der Daten-Struktur der eingelegten Diskette irgend etwas nicht in Ordnung ist. Hierbei wird man auch gleich an den DiskDoctor überwie-

sen. Bei dem DiskDoctor handelt es sich um ein kleines Programm, das versucht, wieder Ordnung in das gefundene Chaos auf der Diskette zu bringen. Das Programm wird für das Laufwerk Null folgendermaßen aufgerufen:

```
diskdoctor drive df0:
```

Anschließend wird man aufgefordert, den "Patienten" in das Laufwerk einzulegen und Return zu drücken. In den folgenden Minuten kann man nur hoffen.

Die Meldungen, die bei der Arbeit mit dem DiskDoctor auftreten können, haben wir im folgenden einmal zusammengestellt und dokumentiert.

Disk Doctor cannot be run in the background

Ursache: Es wurde versucht, dem DiskDoctor durch den vorangestellten Befehl Run einen eigenen Hintergrundprozeß einzuräumen. Der DiskDoctor gehört jedoch zu den wenigen Befehlen, die dies nicht zulassen und nur direkt (ohne Run) gestartet werden können.

Unknown device xxx

Ursache: Vermutlich haben Sie sich bei der Angabe des Gerätenamens verschrieben, da das DOS ein solches Laufwerk nicht kennt (xxx steht hierbei für einen Device-Namen).

Not enough memory

Der DiskDoctor konnte zwar geladen werden, er benötigt aber für seine Funktion mehr Arbeitsspeicher als das System derzeit zur Verfügung stellen kann. Abhilfe: Schließen Sie unbenötigte Fenster und/oder beenden Sie laufende Programme. Seltsamerweise wird diese Fehlermeldung auch dann ausgegeben, wenn man versucht, den DiskDoctor nicht auf eine Diskette, sondern auf ein beliebiges anderes Gerät (Drucker, serielle Schnittstelle etc.) anzuwenden. Solch ein Versuch scheitert natürlich aus ganz anderen Gründen als an einem Mangel an Speicherplatz.

Device xxx not found

Der DiskDoctor kann das dem Gerät zugeordnete Treiber-Device nicht finden. Dieser Fehler wird bei den normalen 3.5-Zoll-Laufwerken wohl nie auftreten, da sich das für ihre Funktion notwendige trackdisk.device fest im ROM (bzw. beim Amiga 1000 im WOM) befindet. Bei Exoten-Laufwerken (5¼ Zoll) ist dieser Fehler meist auf eine fehlerhafte Eintragung des Device-Namens in der sog. Mount-List zurückzuführen. Bei der Verwendung spezieller Disk-Treiber tritt die Fehlermeldung auch dann auf, wenn das in der Mount-List bezeichnete Device nicht gefunden werden kann. Die Aufgabe der Mount-List wird bei der Behandlung des Mount-Befehls noch ausführlich besprochen.

Unable to open disk device

Das Disk-Device ist zwar gefunden worden, kann aber (aus welchen Gründen auch immer) nicht geöffnet werden.

Unexpected end of file

Das DOS verwaltet die Dateien mit einer sehr hohen Redundanz. Viele Informationen sind also öfter vorhanden, als es für eine korrekte Organisation tatsächlich notwendig wäre. Der große Vorteil dieser Redundanz liegt nun darin, daß häufig auch bei teilweisem Datenverlust eine Rekonstruktion der beschädigten Dateien noch möglich ist. Beim Auftreten dieser speziellen Fehlermeldung ist nun der Fall eingetreten, daß sich die an mehreren Stellen der File-Struktur vorhanden Informationen über die Länge der Datei einander widersprechen: Die Datei ist kürzer, als es in dem sog. File-Header angegeben ist.

Error: Unable to access disk

Tritt auf, falls sich das betreffende Laufwerk nicht ansprechen läßt, da zum Beispiel keine Diskette eingelegt worden ist.

Disk must be write enabled

Der Schreibschutz-Clip verhindert zur Zeit einen Schreibzugriff auf die Diskette. Da der DiscDoctor natürlich auch auf die Diskette schreiben will, muß der kleine Schieber an der Diskette so stehen, daß er die Öffnung verdeckt.

Unable to read disk type - formatting track zero

Der DiskDoctor kann den Disketten-Typ (DOS/KICK/BAD), der sich auf Spur Null in dem Sektor Null befindet, nicht lesen. Aus diesem Grunde versucht er nun, die Spur Null neu zu formatieren.

Track zero failed to format - Sorry!

Beim Auftreten dieser Meldung besitzt Ihre Diskette mit großer Wahrscheinlichkeit einen Defekt auf der Spur Null (die am weitesten außen liegende Spur). Falls diese Spur auch bei anderen Disketten häufiger Schwierigkeiten bereitet, kann ein Fehler am Laufwerk vorliegen (Schreib-/Lesekopf positioniert falsch).

Unable to write to root - formatting root track

Der DiskDoctor kann den Track nicht zurückschreiben, auf dem sich der Root-Block befindet. Dieser Root-Block stellt die Wurzel des gesamten Diskettenverzeichnisses dar und ist daher dringend erforderlich. Es wird nun versucht, den betreffenden Track (Track 40, Seite 0) neu zu formatieren und zu initialisieren. Da sich auf diesem Track auch der Name der Diskette befindet, bekommt sie nun den neuen Namen Lazarus.

Root track failed to format - Sorry!

Der Root-Track konnte nicht formatiert werden. Damit kann die Diskette nicht mehr gerettet werden.

Cannot write root block - Sorry!

Der Root-Block kann nicht geschrieben werden. Auch in diesem Fall kann der DiskDoctor nicht mehr helfen.

Warning: File xxx contains unreadable data

Die angegebene Datei (xxx) konnte nicht vollständig rekonstruiert werden und enthält nun nicht lesbare Daten. Eventuell kann hier unter Verwendung eines Disk-Monitors noch etwas gerettet werden. In den meisten Fällen wird man eine solche Datei aber bei der später erscheinenden Abfrage "Delete corrupt files in directory yyy?" durch die Eingabe von YES zum Löschen freigeben müssen.

ATTENTION: Some file in directory xx is unreadable and has been deleted

Hier hat der DiskDoctor die Initiative ergriffen und eine Datei selbständig gelöscht, da für ihre Rekonstruktion zu viele Informationen fehlen.

Failed to read key

Ein Block kann nicht gelesen werden.

Failed to rewrite key

Ein Block kann nicht zurückgeschrieben werden.

Warning: Loop detected at file xx

Eine Datei besteht normalerweise aus einzelnen Blöcken, die miteinander über Blockzeiger wie eine Kette verbunden sind. Die Fehlermeldung besagt nun, daß in der angegebenen Datei eine geschlossene Schleife bei der Verkettung vorhanden ist. Einer der Blöcke der Datei verweist also zurück auf einen schon gelesenen Block. Dadurch kann es passieren, daß ein Lesevorgang der Datei niemals beendet wird, da immer wieder die gleichen Daten gelesen werden.

Parent of key xx is yy which is invalid

Es ist ein Block gefunden worden, der nicht in eine verkettete Liste eingebunden werden kann, da sein Vorgänger-Block ungültig ist.

Hard error Track xx

Der Track mit der Nummer xx kann nicht gelesen werden, da er entweder falsch formatiert ist oder einen echten mechanischen Schaden aufweist. Sehr wahrscheinlich wird es dadurch zu Problemen bei der Rekonstruktion einiger Dateien bzw. Verzeichnisse kommen.

Key xx now unreadable

Der Block mit der Nummer xx ist nicht mehr lesbar.

Replacing dir xx

Das angegebene Verzeichnis konnte rekonstruiert werden und wird nun wieder in die urpsrüngliche Verzeichnisstruktur der Disk integriert.

Inserting dir xx

Das angegebene Verzeichnis konnte rekonstruiert werden und wird nun in das Hauptverzeichnis der Disk eingetragen.

Replacing file xx

Die angegebene Datei konnte rekonstruiert werden und wird nun in das ursprüngliche Verzeichnis eingetragen.

Inserting file xx

Die angegebene Datei konnte rekonstruiert werden und wird nun in das Hauptverzeichnis der Disk eingetragen.

Now copy files to a new disk and reformat this disk

Dies ist die Abschlußmeldung des DiskDoctors. Alle geretteten Dateien und Verzeichnisse können nun in den meisten Fällen auf eine neue Diskette kopiert werden. Anschließend sollte man in jedem Fall die defekte Diskette neu formatieren.

Falls Sie nach dieser letzten Meldung zu voreilig die Diskette aus dem Schacht ziehen, können Sie den Doktor mit großer Wahr-

scheinlichkeit gleich noch einmal beehren, da noch einige Diskettenoperationen ablaufen, die nicht einfach unterbrochen werden dürfen.

Der Befehl unter Workbench 1.3

Der DiskDoctor kann auch auf die resetfeste RAM-Disk RAD angewendet werden.

An der Bedienung des DiskDoctors hat sich nichts geändert. Das Programm hat sich jedoch in der Funktion ein wenig verbessert.

2.1.21 DiskChange

Syntax: `diskchange DEV/A`

Mit diesem sehr lästigen Befehl haben es zum Glück nur die Besitzer eines Laufwerkes für die großen 5¼-Zoll-Disketten zu tun. Diese Stationen teilen - im Gegensatz zu den üblichen 3½-Zoll-Laufwerken - ein Wechseln der Diskette nicht automatisch dem DOS mit. In einem solchen Fall muß also jedesmal DiskChange eingegeben werden, gefolgt von dem Namen, unter dem das Device geführt wird. Erst danach dürfen Zugriffe auf die neue Diskette erfolgen.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2 Systemkommandos des CLI

In den nun folgenden Abschnitten werden die Befehle behandelt, die sich unter dem Oberbegriff Systemkommandos zusammenfassen lassen. Darunter fallen auch die Kommandos, die das CLI betreffen.

2.2.1 NewCLI

Syntax: `newcli WINDOW,FROM`

Der Befehl NewCLI eröffnet dem Benutzer die vielfältigen Möglichkeiten, die das Multitasking des Amiga bietet. Wie Sie wahrscheinlich schon längst wissen, versteht man unter Multitasking die fast gleichzeitige Verarbeitung mehrerer Programme. So ist es zum Beispiel möglich, einen Brief ausdrucken zu lassen, während man gleichzeitig eine Diskette neu formatiert. Natürlich laufen die Befehle nur für einen "zeitlich makroskopischen" Beobachter wirklich nebeneinander ab (daher der Zusatz fast). Als gutes Beispiel sei ein Vergleich mit den Leuchtziffern eines Radioweckers erlaubt:

Nur Eingeweihte wissen, daß in Wirklichkeit alle Ziffern der Uhr nacheinander aufleuchten, was man jedoch durch die hohe Geschwindigkeit, mit der dies geschieht, gar nicht bemerkt. Wenn man das Beispiel weitertreibt, könnte man jede Ziffer der Uhr mit einem Prozeß vergleichen, der vom System (= Uhrbaustein) nur eine bestimmte Rechenzeit (= Aufleuchten) zugewiesen bekommt.

Mit dem NewCLI-Befehl ist es möglich, auf dem Amiga einen solchen Prozeß neu hinzuzufügen. Nach der Eingabe des Kommandos erscheint ein weiteres CLI-Fenster, in dem uns auch gleich mitgeteilt wird, der wievielte Prozeß dies nun schon ist (z.B.: *NewCLI task 2*). Es hindert uns keiner daran, noch weitere CLI-Fenster zu öffnen.

Worin liegen denn die Vorteile mehrerer CLI-Fenster? Man kann ja doch nur in einem Fenster arbeiten. Das stimmt! Man soll ja auch nicht "multitasken", sondern nur der Rechner. Sie können zum Beispiel im ursprünglichen CLI den Befehl *Format Drive df0: NAME Leer* eingeben, anschließend in das neue Fenster überwechseln (einfach anklicken!) und sich hier in der Zwischenzeit das Inhaltsverzeichnis der Diskette ansehen, die sich im zweiten Laufwerk befindet. Einen kleinen Wermutstropfen hat die ganze Angelegenheit natürlich auch: Mit jedem

zusätzlich ablaufenden Prozeß steigt die Bearbeitungsdauer eines Befehls deutlich an.

Wer sich mehr über die Möglichkeiten des Multitasking informieren möchte, der findet in Kapitel 6 eine Fülle von Informationen zu diesem Thema.

Zuletzt noch ein Hinweis auf die möglichen Parameter, die dem NewCLI-Befehl mitgegeben werden können:

Erst einmal kann NewCLI noch um Angaben über die Größe und den Titel des neuen Fensters erweitert werden. Der Befehl

```
newcli con:50/70/250/150/Amiga
```

erzeugt zum Beipiel ein CLI-Fenster Amiga mit der Breite 250 und Höhe 150 (jeweils in Pixeln). Die obere linke Ecke des Fensters liegt an der X-Position 50 und der Y-Position 70.

Das größtmögliche CLI-Fenster erscheint bei der Eingabe des folgenden Befehls:

```
newcli con:0/0/640/256/Amiga
```

Für die direkte Eingabe der Fenstergröße ist diese Option jedoch weniger gedacht, da jedes CLI-Fenster bis auf das Schließsymbol mit allen Manipulations-Gadgets ausgestattet ist. Mit diesen Gadgets kann das Fenster jederzeit auf die gewünschte Größe gebracht werden. Wirklich sinnvolle Anwendungen für die con-Angabe findet man in Verbindung mit den sog. Batch-Dateien, die im Kapitel 5 ausführlich besprochen werden.

Fehlt die Fensterangabe, erhält das CLI immer ein Fenster mit der vollen Breite und knapp der halben Bildschirmhöhe.

Des weiteren kann NewCLI noch um den Zusatz *from* und den Namen einer Batch-Datei erweitert werden, die beim Aufruf des neuen CLI automatisch ausgeführt werden soll. Die Batch-Datei muß sich in dem Ordner befinden, auf den das Verzeichnis S: zeigt. Welcher Ordner dies ist, kann leicht durch den Assign-Befehl ermittelt werden. Beispiel:

```
newcli from Kopieren
```

Bevor Sie mit diesem neuen CLI arbeiten können, wird zuerst noch die Batch-Datei mit dem Namen Kopieren ausgeführt.

Der Befehl unter Workbench 1.3

Der NewCLI-Befehl arbeitet jetzt bei jedem Aufruf die Batch-Datei CLI-Startup ab, die sich im s-Verzeichnis der Workbench-Diskette befindet. In dieser Datei befindet sich normalerweise nur der Prompt-Befehl, der das DOS-Prompt für den neuen CLI-Prozeß festlegt.

Die Zeiten des NewCLI-Befehls sind aber eigentlich vorbei. Im C-Verzeichnis der neuen Workbench-Diskette findet man den neuen Befehl NewShell. Mit diesem Kommando kann eine Fensterschnittstelle zum DOS erzeugt werden, die gegenüber dem CLI einige Vorzüge aufzuweisen hat.

Viele dieser Neuerungen können jedoch nur dann genutzt werden, wenn vor dem Aufruf des NewShell-Befehls das sogenannte Shell-Segment resident im Arbeitsspeicher des Amiga vorhanden ist. Die Befehlsfolge, die hierzu notwendig ist, lautet:

```
resident CLI l:Shell-Seg SYSTEM pure
```

Dieser Befehl wird bei jedem Neustart des Rechners innerhalb der Startup-Sequence automatisch ausgeführt, so daß man sich normalerweise nicht darum kümmern muß. Alle von nun an aufgerufenen Shell-Fenster besitzen die folgenden Eigenschaften:

Residente Befehle werden unterstützt

DOS bietet jetzt die Möglichkeit, die meisten CLI-Befehle resident in den Arbeitsspeicher des Amiga zu laden. Solche Befehle stehen dann dem Anwender jederzeit sofort zur Verfügung. Der Befehl, mit dem dies bewerkstelligt werden kann, lautet Resident. Er wird bei der Behandlung der neuen CLI-Befehle in Kapitel 3 ausführlich besprochen. Der Aufruf dieser residenten Befehle ist nur über die Shell möglich. In einem gewöhnlichen

CLI-Fenster würde ein solcher Befehl weiterhin von der Diskette geladen.

Für alle Befehle können Synonyme eingeführt werden

Wohl manch einer ist schon einmal auf die Idee gekommen, seinen CLI-Befehlen mittels Rename einen kürzeren Namen zu geben. Welche negativen Konsequenzen dies auf dem Amiga mit sich bringt, können Sie sehr leicht feststellen. Benennen Sie hierzu einfach den Befehl Fault, der sich im C-Verzeichnis der Workbench-Diskette befindet, um in FT (rename c:fault as c:ft). Mit Hilfe des Befehles Fault kann man sich den zu einer Fehlermeldung gehörenden Text ausgeben lassen. Selbstverständlich funktioniert diese Funktion auch jetzt noch unter ihrem neuen Namen FT. Gibt man zu Beispiel ein: *ft 103*, so erhält man die Antwort: "Fault 103: insufficient free store".

Versuchen Sie aber nun einmal, mit Hilfe des Kommandos *delete c* das Verzeichnis ihrer CLI-Befehle zu löschen. Natürlich wird Ihnen dies nicht gelingen, da das Verzeichnis nicht leer ist. Anstelle der Fehlermeldung "Not Deleted - directory not empty" erscheint nun aber lediglich die Meldung "Not Deleted - Error code 216". Nun wissen jedoch nur Sie, daß man sich den Text zu dieser Fehlernummer über den FT-Befehl ausgeben lassen kann. Man erkennt daran, daß auch das DOS die CLI-Befehle benutzt. Aus diesem Grund dürfen sie nicht einfach umbenannt werden.

Die neue Shell bietet nun die Möglichkeit, jeden Befehl auch unter einem beliebigen anderen Namen aufzurufen. Die Anweisung hierzu lautet:

```
alias Wunschname Originalname
```

Wunschname steht hierbei für eine beliebige Zeichenkette (ohne Leerzeichen), die den zusätzlichen Namen angibt, unter dem der Befehl aufgerufen werden kann. Der Originalname bezeichnet den Befehl, der bei Eingabe des Wunschnamens ausgeführt werden soll. Findet die Shell zu Beginn der Zeile einen Namen, für den eine solche Zuordnung existiert, so wird dieser Name durch den ihm zugeordneten Befehl ersetzt. Alle anderen Angaben (z.B. zusätzliche Optionen) bleiben unverändert. Beispiel:

```
alias f fault
```

Der Fault-Befehl kann anschließend auch durch die Eingabe eines f, gefolgt von einer Fehlernummer, aufgerufen werden. Trotzdem funktioniert die Ausgabe der Fehlermeldungen im Klartext (vgl. obiges Beispiel) weiterhin, da keine Änderungen auf der Diskette vorgenommen worden sind. Die Zuordnung ist lediglich in eine Tabelle übernommen worden, die von der Shell verwaltet wird.

Die Angabe des Original-Befehls ist übrigens nicht auf ein einzelnes Wort beschränkt. Sollten Sie zum Beispiel einen CLI-Befehl häufig mit den gleichen Optionen aufrufen, können Sie hierfür mittels *alias* einen eigenen Befehl erstellen:

```
alias s-up run ed s:startup-sequence
```

Nun kann man sich durch den Befehl s-up im Nu die Startup-Sequence in den ED holen und sie bei Bedarf modifizieren.

Leider gehen die Namenszuordnungen beim Abstellen des Rechners verloren. Aus diesem Grund hat man eine Batch-Datei eingeführt, die bei jedem Aufruf des NewCLI-Befehls automatisch ausgeführt wird und in die eine beliebige Anzahl alias-Zuordnungen eingetragen werden kann. Diese Datei befindet sich in dem Verzeichnis S: der Workbench-Diskette und trägt den Namen CLI-Startup. Alle hier eingetragenen Zuordnungen sind in jeder Shell gültig.

In dieser Batch-Datei findet man auch eine Zuordnung *alias newshell newcli newcon:0/10/640/100/AmigaShell.* Ein Fenster mit der Überschrift AmigaShell ist also nichts anderes als eine Shell (falls das Shell-Segment resident ist), die ihre Ein- und Ausgaben über das neue Device NewCon: leitet. Dieses Device ist dafür verantwortlich, daß man in der Shell die Zeilen edieren und eine begrenzte Anzahl Befehle zurückholen kann. Das NewCon-Device wird im Kapitel 4 ausführlich behandelt.

Gibt man in der Shell nur das Wort *alias* ein, so erhält man eine Auflistung der derzeitigen Zuordnungen.

Ausgabe des aktuellen Verzeichnis-Pfades

In der neuen Shell besteht das DOS-Prompt nun zusätzlich noch aus der Angabe des aktuellen Verzeichnis-Pfades. Dadurch ist man jederzeit informiert, auf welchem Ast des Verzeichnis-Baumes man sich derzeit befindet. Die lästige Abfrage durch den Befehl CD kann entfallen. Wie man ein eigenes Prompt definieren kann, wird bei der Behandlung des Prompt-Befehls gezeigt.

Direkte Aufrufmöglichkeit einer Batch-Datei

Normalerweise können vom CLI (bzw. einer Shell) aus nur echte Object-Programme gestartet werden. Versucht man zum Beispiel, eine Batch-Datei durch die Eingabe ihres Namens zu starten, so antwortet der Amiga darauf mit der Fehlermeldung: "Unable to load xxx: file is not an object module" (xxx steht hierbei für den File-Namen). Solche Batch-Dateien können nur über den Befehl Execute ausgeführt werden.

Mit Hilfe des neuen Script-Flags ist es jetzt aber auch möglich, eine Batch-Datei ohne Execute aufzurufen. Das DOS erkennt an dem Flag, daß es sich um einen Batch-Datei handelt und ruft automatisch den Execute-Befehl auf. Der Befehl zum Setzen des Flags lautet: protect dateiname +s (siehe Protect-Befehl).

Anstelle des Batch-Files CLI-Startup wird beim Aufruf von NewShell ein Batch-File mit dem Namen Shell-Startup ausgeführt. Auch diese Datei befindet sich im s-Verzeichnis der Workbench-Diskette.

Weitere Informationen zu dem NewShell-Befehl finden Sie im Kapitel 3.

2.2.2 EndCLI

Syntax: `endcli`

Zu dem Befehlsnamen braucht eigentlich schon fast nichts mehr hinzugefügt zu werden. EndCLI beendet einen von der Workbench oder mit NewCLI gestarteten Fenster-CLI-Prozeß wieder. Dabei kann jedoch nur das gerade aktuelle CLI entfernt werden. Man kann also nicht vom ersten CLI aus das Fenster des zweiten CLI schließen. Beendet man ein CLI, von dem aus ein Run-Prozeß (siehe 2.2.3) gestartet worden ist, der noch aktiv ist, wird der CLI-Prozeß zwar beendet, das Fenster bleibt aber noch für eventuelle Ausgaben des Run-Prozesses geöffnet. Erst wenn der letzte Prozeß beendet ist, verschwindet auch das Fenster.

Vorsicht ist beim letzten CLI geboten: Ist die Workbench nicht geladen und gibt man nun *Endcli* ein, so hat man sich soeben den Ast abgesägt, auf dem man sitzt. Wie soll man dem DOS nun mitteilen, daß man irgendein Programm starten möchte?

Der Befehl unter Workbench 1.3

Mit EndCLI kann auch die Arbeit mit einer Shell beendet werden. Einen Befehl ENDSHELL gibt es nicht. Oft befindet sich aber in dem Batch-File Shell-Startup eine Alias-Zuordnung der Form *alias endshell endcli*, so daß in einer Shell auch die Eingabe von *endshell* akzeptiert wird.

2.2.3 Run

Syntax: `run PROGRAMMNAME`

Neben der schon besprochenen Möglichkeit, mit NewCLI einen weiteren Prozeß zu erzeugen, kann dies auch von nur einem CLI aus erreicht werden. Jedes Programm (und damit auch jeder CLI-Befehl) kann, wenn er durch Run gestartet wird, als sogenannter Hintergrundprozeß ablaufen. Gleichzeitig kann nun mit demselben CLI weitergearbeitet werden. Eventuelle Ausgaben

des Run-Prozesses landen (falls nicht anders angegeben) im CLI-Fenster, von dem aus der Prozeß gestartet wurde. Beispiel:

```
run c/join Brief1 Brief2 Brief3 to prt:
dir ram:
```

Hier haben wir es gleich mit einer Kombination sinnvoller Anwendungen von CLI-Befehlen zu tun: Der Join-Befehl wird zweckentfremdet und für die Ausgabe mehrerer Briefe auf den Drucker benutzt. Damit man hierbei nicht warten muß, wird für diese Aufgabe durch Run ein Hintergrundprozeß erzeugt. Während der Drucker nun loslegt, kann man sich im CLI das Inhaltsverzeichnis der RAM-Disk ansehen.

Was wäre aber gewesen, wenn es den Join-Befehl nicht gegeben hätte? Müßten dann jeweils die Briefe einzeln durch *run type Briefx to prt:* ausgegeben werden?

Nein, für die Bearbeitung mehrerer Befehle hintereinander durch Run existiert das Pluszeichen. Die folgenden Zeilen sind somit äquivalent zum obigen Join-Befehl:

```
run type Brief1 to prt: +
type Brief2 to prt: +
type Brief3 to prt:
```

Der gesamte Block wird erst dann als Hintergrundprozeß abgearbeitet, wenn nach Eingabe der letzten Zeile (ohne +) die Return-Taste betätigt wird.

Der Run-Befehl der Workbench Version 1.2 sollte unter keinen Umständen umbenannt werden, da er auch von anderen CLI-Befehlen aus aufgerufen wird. NewCLI funktioniert zum Beispiel nur in Verbindung mit Run. Es genügt daher auch nicht, nur den NewCLI-Befehl in die RAM-Disk zu kopieren, falls man jederzeit einen schnellen Zugriff auf diesen Befehl haben möchte.

Der Befehl unter Workbench 1.3

Werden die Ausgaben eines durch Run gestarteten Prozesses direkt auf ein beliebiges Gerät umgeleitet, so soll es nun auch möglich sein, die Shell, von der aus der Prozeß gestartet worden ist, durch *endcli* zu verlassen, ohne daß das zugehörige Fenster noch für eventuelle Ausgaben geöffnet bleibt.

Der nun folgende Befehl erzeugt einen Hintergrundprozeß, der das gesamte Inhaltsverzeichnis der Diskette in Laufwerk Null in eine Datei mit dem Namen Liste schreiben soll:

```
run >Liste dir df0: opt a
```

Theoretisch soll es nun möglich sein, mit EndCLI die Shell zu verlassen und dadurch das Fenster zu schließen, während der Dir-Befehl im Hintergrund weiterarbeitet. Leider funktioniert dieses Verfahren derzeit noch nicht, da das Gerät, auf das die Ausgabe umgelenkt wird, ein End-Of-File-Steuerzeichen bekommt, noch ehe überhaupt ein Zeichen des gestarteten Befehls übetragen werden kann. Lediglich die Nummer des Prozesses (z.B. *CLI [2]*) wird an das Gerät übergeben.

Sieht man sich unsere Beispieldatei einmal mit Hilfe des Befehls *type Liste* an, so erscheint anstelle des gewünschten Disketten-Verzeichnisses lediglich die Prozeßnummer.

2.2.4 Status

Syntax: `status PROCESS,FULL/S,TCB/S,CLI=ALL/S`

Mit dem Befehl Status kann man sich jederzeit über die derzeit laufenden CLI-Tasks informieren. Gibt man den Befehl ohne weitere Parameter oder gefolgt von CLI bzw. ALL ein, erhält man die Namen der einzelnen Tasks. Beispiel:

```
Task  1: Loaded as command: status
Task  2: Loaded as command: textomat
```

Da das Textomat-Programm vom CLI aus durch Run gestartet worden ist, hat es die Task-Nummer Zwei bekommen. Das CLI ist natürlich gerade mit dem Status-Befehl beschäftigt.

Ferner gibt es noch die Möglichkeit, sich nur die Informationen zu einer speziellen Task ausgeben zu lassen. Hierzu muß der Status-Befehl noch um die zugehörige Prozeß-Nummer ergänzt werden: Status 2 würde nur die zweite Zeile des obigen Beispiels zeigen.

Erweitert man den Status-Befehl um die Buchstabenkombination TCB, so bekommt man noch weitere Informationen zu den einzelnen Tasks. In unserem Beipiel sieht das so aus:

```
Task  1: stk 1600, gv 150, pri 0
Task  2: stk 3200, gv 150, pri 0
```

Die Angaben hinter der Task-Nummer haben die folgende Bedeutung:

stk

Die hier eingetragene Zahl gibt die Größe des Prozessor-Stacks für diese Task an.

gv

Umfang der Global-Vector-Tabelle.

pri

Dieser Wert gibt die Priorität der jeweiligen Tasks wieder. Die möglichen Werte reichen theoretisch von -128 bis +127. Negative Prioritäten werden jedoch fälschlicherweise durch ihr 2er-Komplement dargestellt. Dadurch bekommen auch sie ein positives Vorzeichen. Weiß man jedoch, daß eine Priorität eigentlich negativ sein müßte, so kann man durch die Subtraktion von 256 die tatsächliche Priorität dieser Tasks bestimmen.

Mehr Informationen zu den beiden letzten Werten finden Sie in den Kapiteln 6 und 7.

Eine Zusammenfassung aller Informationen zu den Tasks erhält man schließlich durch den Zusatz FULL. Das Kommando *status full* ergibt für das obige Beispiel:

```
Task  1: stk 1600, gv 150, pri 0 Loaded as command: status
Task  2: stk 3200, gv 150, pri 0 Loaded as command: textomat
```

Der Befehl unter Workbench 1.3

Syntax: `PROCESS,FULL/S,TCB/S,CLI=ALL/S,COM=COMMAND/K`

Der neue Status-Befehl gibt auch negative Prioritäten korrekt wieder. Außerdem ist die neue Option COM=COMMAND/K hinzugekommen. Mit ihrer Hilfe kann getestet werden, ob sich ein bestimmtes Programm unter den derzeit vorhandenen Prozessen befindet. Dazu muß der Status-Befehl durch den Zusatz COM (bzw. COMMAND) und den Namen des Prozesses ergänzt werden. Beispiel:

```
status com textomat
```

Falls ein Prozeß mit dem Namen textomat gefunden wird, gibt der Status-Befehl die zugehörige Prozeß-Nummer aus. Andernfalls erfolgt keine Ausgabe, und es wird der Error-Code 5 zurückgeliefert. Diese Option eignet sich daher besonders für die Verwendung in Batch-Dateien. Man kann durch eine solche Abfrage zum Beispiel vehindern, daß ein bestimmter Hintergrund-Prozeß mehrmals gestartet wird.

2.2.5 ChangeTaskPri

Syntax: `changetaskpri PRI/A`

Mit Hilfe des Kommandos ChangeTaskPri kann für das derzeitige CLI eine neue Priorität festgelegt werden. Eine ausführliche Beschreibung, was es mit den Prioritäten auf sich hat, finden Sie im Kapitel 6 dieses Buches. An dieser Stelle soll die folgende (stark vereinfachte) Darstellung genügen:

Jede Task im Amiga hat eine bestimmte Dringlichkeitsstufe, im folgenden Priorität genannt. Wie Sie bei dem letzten Kommando

Status schon erfahren haben, kann dieser Wert zwischen -128 und +127 liegen. Das Kommando ChangeTaskPri erlaubt es nun, den aktuellen CLI-Prozeß (und nur diesen!) in seiner Priorität zu beeinflussen. Um zum Beispiel die Priorität auf +5 zu setzen, genügt die folgende Eingabe:

```
changetaskpri 5
```

Ein anschließendes *status full* ergibt:

```
Task  1: stk 1600, gv 150, pri 5 Loaded as command: status
Task  2: stk 3200, gv 150, pri 0 Loaded as command: textomat
```

Sollte die Eingabe außerhalb des erlaubten Bereichs liegen, quittiert der Befehl dies durch die Fehlermeldung:

```
Priority out of range (-128 to +127)
```

Der Befehl unter Workbench 1.3

Syntax: `changetaskpri PRI/A,PROCESS/K`

Neu hinzugekommen ist die Option Process/k. Damit ist es nun auch möglich, die Priorität jedes beliebigen Prozesses zu beeinflussen. Hinter dem Schlüsselwort Process muß dazu die zugehörige Prozeß-Nummer angegeben werden. Diese Nummer kann man mit Hilfe des Status-Befehls erhalten. Beispiel:

```
changetaskpri Pri -5 Process 4
```

Der Prozeß mit der Nummer 4 bekommt die neue Priorität -5 zugewiesen.

Sehr nützlich ist diese neue Option, falls man zum Beispiel einen umfangreichen Druckvorgang als Hintergrund-Prozeß gestartet hat und nun feststellt, daß dieser Prozeß zuviel Rechenzeit in Anspruch nimmt und sich dadurch die Arbeitgeschwindigkeit anderer Prozesse (z.B. einer Textverarbeitung) auf ein unerträgliches Maß reduziert. Hier kann man nun mit Hilfe des Befehls ChangeTaskPri die Priorität des Druck-Prozesses nachträglich z.B. auf minus 5 setzen. Dadurch wird in den meisten Fällen den übrigen Tasks wieder genügend Zeit zur Verfügung gestellt. Daß der Druckvorgang jetzt natürlich etwas länger dauert, wird kaum

stören, da ja in der Zwischenzeit beispielsweise in der Textverarbeitung weitergearbeitet werden kann.

2.2.6 Break

Syntax: `break PROCESS/A,ALL/S,C/S,D/S,E/S,F/S`

Daß man mit Crtl C ein gerade ablaufendes CLI-Kommando unterbrechen kann, ist Ihnen sicher längst bekannt. Weit weniger bekannt ist da schon die Möglichkeit, dasselbe auch von einem anderen CLI-Fenster aus softwaremäßig zu erreichen. Genau hierfür existiert der Befehl Break. Er erlaubt es, von jedem beliebigen CLI-Fenster aus die Abarbeitung eines DOS-Befehls in einem anderen Fenster unterbrechen zu können. Gibt man beispielsweise im ersten Fenster den Befehl *dir opt a* ein, kann man die Ausgabe des vollständigen Inhaltsverzeichnisses vom zweiten CLI-Fenster aus durch die Eingabe von *break 1* unterbrechen. Wo liegt nun der spezielle Nutzen dieses Befehls?

Nun, in diesem Fall könnte man dasselbe Ergebnis erzielen, indem man mit der Maus das erste Fenster aktiviert und hier die Tastenkombination Ctrl und C eingibt. Auch hierdurch wird der Dir-Befehl unterbrochen. Es gibt aber auch sinnvollere Anwendungen für diesen Befehl. Können Sie sich noch an das erste Beispiel für den Run-Befehl erinnern? Dort wurde für die Ausgabe mehrerer Briefe auf den Drucker ein eigener Prozeß gestartet. Was soll man aber machen, wenn man den Druckvorgang vorzeitig abbrechen möchte? Einfach den Drucker abzustellen, ist bestimmt nicht die feine Art, und die Eingabe von Ctrl und C in dem Fenster, von dem aus der Prozeß gestartet worden ist, bringt gar nichts, da sich der Prozeß von den CLI-Eingaben völlig getrennt hat. In dieser Situation kann nur der Break-Befehl helfen: *break 2* läßt das Druckergeräusch verstummen. Die Zahl hinter dem Break-Befehl muß immer mit der vom System vergebenen Nummer für den zu unterbrechenden Prozeß übereinstimmen.

Ohne weitere Parameter sendet der Break-Befehl immer ein Ctrl C. Es sind aber auch die Ctrl-Kombinationen D bis F möglich.

Dazu kann der Befehl einfach um die entsprechenden Buchstaben erweitert werden. Zum Beispiel:

```
break 3 d
```

Ein Suchvorgang *search df0: all*, der die Prozeßnummer 3 zugeordnet bekommen hat, würde durch Break 3 d sofort die Durchsuchung des nächsten Files beginnen. Welche Reaktionen durch Ctrl-Kombinationen außer Ctrl und C ausgelöst werden, ist von Fall zu Fall verschieden und hängt vom jeweiligen CLI-Befehl ab. In den meisten Fällen passiert einfach gar nichts.

Schließlich hat man noch die Möglichkeit, alle Ctrl-Kombinationen auf einmal an einen Prozeß zu schicken. Hierzu ist nur der Zusatz *all* hinter der Prozeßnummer einzutragen.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.7 Path

Syntax: `path ,,,,,,,,,,ADD/S,SHOW/S,RESET/S`

Gibt man den Befehl Path ohne Parameter oder gefolgt von *show* ein, antwortet der Rechner mit der Ausgabe einer (mehr oder weniger langen) Liste. Diese könnte zum Beispiel folgendermaßen aussehen:

```
Current directory
RAM:c
A500 WB 1.2 D:System
C:
```

Diese Liste gibt an, in welcher Reihenfolge und in welchen Verzeichnissen nach einem File gesucht wird. Gibt man den Namen eines Programms (z.B. den eines CLI-Kommandos) ein, sucht das DOS - wie Sie schon in 2.1 erfahren haben - zuerst einmal im aktuellen Verzeichnis (Current Directory) nach einem solchen File. Welches Directory zur Zeit current ist, kann jederzeit durch das Kommando *CD* in Erfahrung gebracht werden.

Sollte DOS hier nicht fündig werden, wird noch das RAM:C-Verzeichnis und danach der Systemordner der Workbench-Diskette durchforstet. Ist die gesuchte Datei immer noch nicht dabei, sucht das DOS zuletzt noch im Gerät C. Wie bitte? Solch ein Gerät haben Sie gar nicht angeschlossen? Wir können Sie beruhigen: wir auch nicht. Es handelt sich dabei um ein logisches Gerät, das nur für den Rechner existiert. Dieses Pseudo-Device ist notwendig, damit die CLI-Kommandos auch im richtigen Verzeichnis gesucht werden. Was es mit diesen logischen Geräten genau auf sich hat, finden Sie bei der Beschreibung des Assign-Befehls.

Der Path-Befehl erlaubt es nun, weitere Pfade hinzuzufügen, oder schon vorhandene zu löschen. Wenn Sie zum Beipiel öfter den Taschenrechner benötigen, der sich auf der Workbench-Diskette im Ordner Utilities befindet, mußten Sie normalerweise folgenden Befehl zum Laden des Programms eingeben: Utilities/ Calculator

Wenn Sie aber vorher das Kommando: *path sys:Utilities add* eingeben, wird die Path-Liste um folgenden Eintrag erweitert:

```
A500 WB 1.2 D:Utilities
```

Nun ist es nicht mehr notwendig, beim Laden des Taschenrechners den Pfad (Utilities) anzugeben, da das DOS automatisch auch in diesem Verzeichnis nachsieht.

Sehr interessant ist der Path-Befehl auch in Verbindung mit der RAM-Disk. Da zusätzliche Pfade in der Liste immer vor dem Eintrag für das C-Device eingefügt werden, ist es möglich, einige CLI-Befehle in die RAM-Disk zu kopieren und einen entsprechenden Pfad zu der Path-Liste hinzuzufügen (path ram: add). Das erspart gerade den Besitzern nur einer Floppy eine Menge Arbeit, da zuerst einmal in der RAM-Disk nachgesehen wird, ob das entsprechende Kommando hier zu finden ist. Nur wenn der Befehl dort nicht vorhanden ist, wird man aufgefordert, die Workbench-Diskette einzulegen. Zu diesem Thema finden Sie auch im Kapitel 4 noch einige Hinweise.

Insgesamt können maximal 10 Pfadangaben gleichzeitig neu hinzugefügt werden. Dazu muß das Wort *add* am Ende der Liste erscheinen. Gibt man jedoch anstelle von *add* die Option *reset* ein, so werden alle Pfade bis auf die maximal 10 angegebenen wieder gelöscht. Der Befehl *path reset* löscht damit alle Pfade, bis auf das aktuelle Verzeichnis (current directory) und das dem Device C zugeordnete Verzeichnis.

Der Befehl unter Workbench 1.3

Die Funktionsweise des Path-Befehls hat sich nicht geändert, obwohl sich die Reihenfolge des Suchvorgangs ein wenig verändert hat. Falls zu einem Befehl kein spezieller Pfad angegeben wird, sucht das DOS ihn zunächst einmal unter den residenten Befehlen. Nur wenn der Befehl nicht resident im Arbeitsspeicher vorhanden ist, wird der Suchvorgang so fortgesetzt, wie es der Path-Befehl angibt (current directory etc.).

2.2.8 Assign

Syntax: `assign NAME,DIR,LIST/S`

Auch bei diesem Befehl schauen wir uns zunächst die Liste an, die der Rechner ausgibt, falls man nur *assign* bzw. *assign list* eingibt:

```
Volumes:
RAM Disk [Mounted]
A500 WB 1.2 D [Mounted]

Directorys:
S               A500 WB 1.2 D:s
L               A500 WB 1.2 D:l
C               A500 WB 1.2 D:c
FONTS           A500 WB 1.2 D:fonts
DEVS            A500 WB 1.2 D:devs
LIBS            A500 WB 1.2 D:libs
SYS             A500 WB 1.2 D:

Devices:
DF1  DF0  PRT  PAR  SER
RAW  CON  RAM
```

Unter der Überschrift Volumes findet man die Namen der Disketten (bzw. bei Festplatten: der Partitions), die das DOS zur Zeit kennt. Der Zusatz [Mounted] bedeutet, daß sich die Diskette derzeit im Laufwerk befindet. Das ist natürlich bei der RAM-Disk nicht ganz wörtlich zu nehmen.

Interessant wird es nun bei den Eintragungen unter Directorys. Auf der linken Seite stehen hier die Namen von Schein-Geräten. Das C haben Sie ja schon bei der Besprechung des Path-Befehles kurz kennengelernt. Jedem dieser Pseudo-Devices ist nun eine vollständige Pfadbeschreibung zu einem tatsächlich existierenden Verzeichnis zugeordnet. Man könnte den Namen eines Pseudo-Device auch als das Synonym für den rechts nebenstehenden Pfad bezeichnen. Jetzt wird auch klar, warum die CLI-Kommandos überhaupt gefunden werden: Dem Device C ist der C-Ordner auf der Workbench-Diskette zugeordnet. In diesem befinden sich bekanntlich alle CLI-Befehle. Der Path-Befehl hatte uns schon gezeigt, daß das C-Device grundsätzlich nach den eingegebenen Befehlen durchsucht wird.

Daß die Device-Namen mit den Ordnernamen übereinstimmen, ist keinesfalls eine Bedingung: Ein Programm, das auf das Device FONTS: zugreift, könnte auch genausogut an ein Verzeichnis mit dem Namen Zeichensätze verwiesen werden.

Natürlich wäre der Befehl Assign sinnlos, wenn man keine Änderungen an den Zuweisungen vornehmen könnte. Hierfür sind die Optionen des Assign-Befehles vorgesehen:

NAME

Steht für einen Device-Namen (Das DOS erkennt diesen an einem Doppelpunkt!).

Dir

Gibt man hier einen vollständigen Pfadnamen an, so wird dem Eintrag unter NAME dieser Pfad zugeordnet. Fehlt diese Angabe, löscht der Befehl das unter Dir angegebene Device aus der Liste.

LIST

Durch diesen Zusatz erzielt man bei einer Änderung eine anschließende Ausgabe der aktuellen Assign-Liste. Werden keine Änderungen vorgenommen, kann man bei dem Assign-Befehl auch auf die Option LIST verzichten.

Ganz am Schluß der Ausgabe findet man unter der Überschrift Devices noch die Namen der echten Geräte, die vom CLI aus angesprochen werden können. Diese Geräte werden im Kapitel 4 ausführlich besprochen. Bei Gerätenamen, die aus mehr als drei Zeichen bestehen, kann es aufgrund einer fehlerhaften Programmierung vorkommen, daß in der Liste Leerzeichen zur Trennung der einzelnen Namen fehlen.

Der Befehl unter Workbench 1.3

Mit Hilfe des Assign-Befehls läßt sich jetzt auch testen, ob ein bestimmtes Device existiert. Dazu muß der Assign-Befehl ergänzt werden durch den entsprechenden Gerätenamen und den Zusatz LIST. Falls das angegebene Gerät vorhanden ist, wird der zugehörige Eintrag ausgegeben und der Fehlerstatus auf Null gesetzt. Beispiel:

```
Eingabe:    assign fonts: list
Ausgabe:    Volumes:

            Directories:
            fonts           Volume: A500 WB 1.2 D

            Devices:
```

Das Gerät fonts: ist hier unter der Überschrift Directories gefunden worden: Es handelt sich offensichtlich um ein Schein-Device.

Falls das gesuchte Gerät nicht gefunden werden kann, gibt der Assign-Befehl den Fehlerstatus 5 zurück. Dieser Fehlerstatus eignet sich besonders zur Auswertung innerhalb einer Batch-Datei (s. Kapitel 5). Die nun folgende kleine Batch-Datei testet, ob sich die Diskette Extras in einem der Laufwerke befindet. Falls nicht, so wird eine Aufforderung an den Benutzer ausgegeben, diese Diskette in ein Laufwerk einzulegen.

```
assign >nil:  Extras: list
if warn
   echo "Bitte legen Sie die Diskette Extras in ein Laufwerk"
endif
```

Durch den Zusatz *>nil:* werden alle Ausgaben des Assign-Befehls auf das Nil-Device umgeleitet. Dieses Gerät übernimmt die Funktion eines Müllschluckers: Die umgeleiteten Daten tauchen nirgends auf. Dadurch können unerwünschte Ausgaben auf eine einfache Art und Weise unterdrückt werden. Lediglich der Fehlerstatus interessiert uns. Dieser kann durch den Befehl *if warn* abgefragt werden. Falls der Assign-Befehl die Extras-Diskette nicht gefunden hat, ist die *if warn*-Bedingung erfüllt (warn = Fehlerstatus 5), und es erscheint der angegebene Text.

2.2.9 AddBuffers

Syntax: `addbuffers DRIVE/A,BUFFERS/A`

Haben Sie es bei der Arbeit im CLI schon einmal erlebt, daß ein Befehl zwar bei der ersten Benutzung vom Laufwerk geladen werden mußte, er aber bei einer erneuten Anwendung sofort zur Verfügung stand? Die Ursache für dieses erfreuliche Phänomen ist bei den Pufferspeichern der Laufwerke zu suchen. In diese werden alle Daten erst einmal geladen, bevor sie weiterverarbeitet werden. Ist ein Programm nur so groß, daß es vollständig in den Puffer hineinpaßt, muß es bei einem erneuten Aufruf nicht wieder von der Diskette oder der Harddisk geladen werden. Hierdurch wird der Ladevorgang natürlich extrem beschleunigt.

Das Kommando AddBuffers erlaubt es nun, einem angeschlossenen Laufwerk einen größeren Pufferspeicher zuzuordnen. Durch die Eingabe *addbuffers df0: 11* zum Beispiel wird dem Laufwerk 0 ein zusätzlicher Pufferspeicher für 11 Blocks eingeräumt (1 Block = 512 Byte). Hierdurch kann zusätzlich einer der insgesamt 160 Tracks einer Diskette vollständig in den Speicher geladen werden.

Wie fast alle schönen Dinge hat auch der Befehl AddBuffers einen kleinen Haken: Der Speicherplatz wird Ihnen natürlich

vom Systemspeicher abgezogen. Insgesamt wird sogar pro Block ein Speicher von 544 Bytes benötigt. Die restlichen 32 Bytes werden intern zur Speicherverwaltung benötigt. Für den Bedarf an Systemspeicher wird man sicherlich Verständnis zeigen, denn irgendwo müssen die Daten schließlich abgelegt werden. Unverständlich hingegen ist, warum man einmal reservierte Pufferspeicher nicht wieder freigeben kann. Dies ist nur durch einem Neustart des Systems möglich. So flexibel wie der Amiga ist, wird es aber bald sicherlich auch eine Option SubBuffers geben.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.10 Why

Syntax: `why`

Die Aufgabe dieses Befehls ist relativ schnell erklärt: Sollte ein CLI-Befehl aus irgendeinem Grund die gewünschte Funktion nicht ausführen, kann man in einigen Fällen mit dem Befehl Why im CLI die Ursache hierfür erfragen. Beispiel: Man möchte sich die Startup-Sequence ansehen und gibt daher ein:

```
type s:startup-sequenz
```

Daraufhin die Reaktion des Rechners:

```
Can't open s:startup-sequenz
```

Fragen wir jetzt mit Why nach der Ursache bekommen wir zur Antwort:

```
Last command failed because object not found
```

Kein Wunder, daß der type-Befehl die Datei nicht gefunden hat, denn wir haben aus Versehen die Startup-Sequence am Ende mit z geschrieben.

Falls der Why-Befehl keine Angaben zur Ursache einer Fehlfunktion machen kann, gibt er die folgende Meldung aus:

```
The last command did not set a return code
```

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.11 Fault

Syntax: `fault ,,,,,,,,`

Dieser Befehl gibt zu den maximal neun angegebenen Fehlernummern die zugehörige Meldung des AmigaDOS aus. Echte Fehlertexte gibt es jedoch nur in einigen Fällen. Sollte ein entsprechender Fehlertext nicht existieren, wird einfach das Wort Error gefolgt von der jeweiligen Nummer ausgegeben. Dazu zwei Beispiele:

```
1. Eingabe:   fault 10 20 25
  Ausgabe:    Fault 10: Error 10
              Fault 20: Error 20
              Fault 25: Error 25
2. Eingabe:   fault 120 121 122 123
              Fault 120: argument line invalid or too long
              Fault 121: file is not an object module
              Fault 122: invalid resident library during load
              Fault 123: Error 123
```

Die Fault-Funktion wird auch von dem im letzten Abschnitt behandelten Why-Befehl aufgerufen. Da das gegenseitige Aufrufen der CLI-Befehle im AmigaDOS verbreitet ist, darf man CLI-Befehle nicht einfach umbenennen. Sollte der Why-Befehl die Fault-Funktion nicht mehr unter ihrem ursprünglichen Namen aufrufen können, wird lediglich die zugehörige Fehlernummer ausgegeben (z.B.: Last command failed because Error code 205).

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.12 Date

Syntax: `TIME,DATE,TO=VER/K`

Dieser Befehl ermöglicht es, unabhängig von dem Programm Preferences die aktuelle Systemzeit (Uhrzeit und Datum) des Amiga jederzeit stellen und auch lesen zu können.

In der Syntax-Liste steht TIME für eine Uhrzeit im Format HH:MM:SS (H = Stunden, M = Minuten, S = Sekunden) oder nur im Format HH:MM.

DATE muß ein Datum im Format DD-MM-YY sein (D = Tag, M = Monat, Y = Jahr). Sollte das alte Datum höchstens eine Woche zurückliegen, kann hier auch einfach der aktuelle Wochentag auf Englisch eingegeben werden. Ebenso ist für das Zurückdatieren um einen Tag das Wort *Yesterday* zugelassen. In beiden Fällen wird das Datum korrekt berechnet.

Mit dem Zusatz TO bzw. VER kann die Ausgabe auf eine Datei umgelenkt werden. Gibt man den Befehl ohne Parameter ein, wird der aktuelle Wochentag, das Datum und die Uhrzeit auf dem Bildschirm angezeigt:

```
Tuesday 17-May-88 20:10:30
```

Wohl die wenigsten unter Ihnen werden sich mit Hilfe des Date-Befehls den Wochentag ihrer Geburt ausrechnen können: Der ewige Kalender funktioniert erst ab dem 2. Januar 1978. Der 1. Januar wird als *unset* angesehen. Zeitpunkte, die davor liegen, sind sogar ungültig (invalid).

Der Befehl unter Workbench 1.3

Der neue Date-Befehl akzeptiert jetzt sowohl eine einstellige als auch eine zweistellige Angabe des aktuellen Tages. Damit ist z.B. anstelle der Eingabe *date 01-jun-88* nun auch die Eingabe *1-jun-88* möglich.

2.2.13 SetClock

Syntax: `setclock opt load|save`

Nur für die Besitzer eines Amiga mit batteriegepufferter Echtzeituhr ist dieser Befehl interessant. Er wird in den meisten Fällen dazu benutzt, die von der Batterie-Uhr bereitgestellten Informationen (Zeit und Datum) nach dem Einschalten des Rechners als neue Werte in die System-Uhr zu übernehmen. Dies ist nach jedem Start des Amiga notwendig, da beide Uhren völlig unabhängig voneinander arbeiten. Dazu muß der Befehl durch den Zusatz *opt load* ergänzt werden.

Außerdem besteht die Möglichkeit, mit Hilfe des Befehls SetClock die derzeitige Systemzeit des Amiga (die mit Date gesetzt werden kann) in die Batterie-Uhr zu übertragen. Diese Funktion wird durch *setclock opt save* aufgerufen.

In den meisten Fällen wird man den Befehl zum Lesen der Batterie-Uhr in der Startup-Sequence der Boot-Diskette wiederfinden. Bei der original Workbench-Diskette steht dort in einer Zeile: *SetClock >nil: opt load*. Es wird also eine auftretende Meldung des Befehls auf das NIL-Device gegeben. Dieses Schein-Device verhindert einfach nur, daß Ausgaben auf dem Bildschirm erscheinen. Stellt der Befehl SetClock nämlich fest, daß überhaupt keine Echtzeituhr installiert ist, meldet er:

```
Internal clock not functioning
```

Damit nun nicht jeder, der eine solche Uhr nicht besitzt, diese Ausgabe erhält, wird die Meldung kurzerhand unterdrückt. Die ganze Prozedur dauert in einem solchen Fall mit dem Laden des Befehls von der Diskette ca. 6 Sekunden und bringt rein gar nichts. Der Befehl kann daher auch ersatzlos aus der Startup-Sequence gestrichen werden.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.14 Prompt

Syntax: `prompt TEXT`

Unter einem Prompt versteht man in der DOS-Welt die Zeichenkombination, die angezeigt wird, falls der Rechner bereit ist, neue Eingaben entgegenzunehmen. Normalerweise besteht er im Amiga aus der Nummer des jeweiligen CLI und einer spitzen Klammer: 1>. Nicht so leistungsfähige Rechner zeigen hier meist an, auf welches Laufwerk sich alle Eingaben beziehen sollen. So etwas Seltsames kennt der Amiga aber nicht!

Der Befehl Prompt hat einen rein kosmetischen Charakter und erlaubt die Eingabe eines beliebigen Textes, der anstelle des alten Prompts erscheinen soll. Enthält dieser Text Leerzeichen, ist er (wie üblich) in Anführungszeichen zu setzen. Beispiel:

```
prompt "Was willst Du? "
```

Anschließend duzt Sie Ihr Rechner. Wird der Befehl ohne einen Text eingegeben, besteht das Prompt nur aus der spitzen Klammer (>). Soll jedoch an einer bestimmten Stelle die Nummer des jeweiligen CLI-Prozesses erscheinen, muß hierfür die Kombination %n eingetragen werden. Beispiel:

```
prompt "Ich bin die Nummer %n ! "
```

Ergebnis: (je nach CLI-Nummer)

```
Ich bin die Nummer 1 !
```

Demnach erreicht man das Standardprompt durch:

```
prompt %n>
```

Der Befehl unter Workbench 1.3

Mit Hilfe des neuen Prompt-Befehls ist es jetzt auch möglich, das jeweils aktuelle Laufwerks-Verzeichnis als Teil des Prompt-Textes ausgeben zu lassen. Zusätzlich zu dem Steuerzeichen %n, mit dem bekanntlich die Nummer des aktuellen CLI-Prozesses angezeigt werden kann, existiert hierfür das Steuerzeichen %s.

Dadurch kann die lästige Orientierung mittels des CD-Befehls endlich entfallen. Beispiel:

```
prompt "%n.%s> "
```

Das neue Prompt könnte anschließend folgendermaßen aussehen:

```
3.Workbench 1.3:System>
```

Man befindet sich hier im dritten CLI-Prozeß. Das aktuelle Verzeichnis ist der System-Ordner der Workbench-Diskette.

2.2.15 Stack

Syntax: `stack SIZE`

Jeder CLI-Prozeß stellt geladenen DOS-Befehlen einen speziellen Speicherbereich zur Verfügung, der von Programmierern einer maschinennahen Sprache Stack (= Stapel) genannt wird. Normalerweise beträgt die Größe eines solchen Bereiches 4000 Bytes pro CLI. Mit Hilfe des Kommandos Stack gefolgt von einer ganzen Zahl, die größer oder gleich dem Mindestwert 1600 sein muß, kann man nun die Größe des Stapelspeichers frei wählen. Dabei muß man aber höllisch aufpassen: Benötigt ein Befehl einen größeren Speicher, als wir dem CLI zugewiesen haben, kann es zu Systemabstürzen kommen, bei denen selbst der Guru nicht mehr dazwischen kommt! Besonders der Dir-Befehl ist hierfür sehr anfällig. Probieren Sie dies ruhig einmal mit der Workbench-Diskette aus, wenn sich gerade nichts Wichtiges in Ihrem Arbeitsspeicher befindet:

```
stack 1600
dir opt a
```

Schon ist nichts mehr zu retten. Auch der Sort-Befehl ist recht anspruchsvoll, was den Bedarf an Stapelspeicher angeht. Dieser hängt maßgeblich vom Umfang der zu sortierenden Datei ab. Fest vorgegebene Werte gibt es aber leider nicht. Hier hilft nur: trial and error.

Interessant ist ferner noch die Tatsache, daß ein neuer Prozeß immer genausoviel Stapelspeicher zugewiesen bekommt, wie das CLI besitzt, von dem aus er gestartet wurde. Beachtet man dies nicht, kann man bei der Arbeit mit mehreren "aufgepumpten" CLI-Stacks sehr schnell viel Speicherplatz verschenken.

Ist man sich nicht mehr darüber im klaren, wie groß der derzeitige Stapelspeicher ist, kann man ihn durch Status (ohne Parameter) jederzeit erfragen.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.16 BindDrivers

Syntax: `binddrivers`

Der Befehl BindDrivers ist nur für die Betreiber außergewöhnlicher Peripheriegeräte (Harddisk, Plotter etc.) von Interesse. Man bindet dieses Kommando dann meist in die Startup-Sequence der Systemdiskette ein. Durch den Befehl BindDrivers wird veranlaßt, daß die Gerätetreiber, die sich im Ordner Expansion befinden, in das System eingebunden werden. Diese Treiber sind im Lieferumfang der Geräte vorhanden und für eine korrekte Funktion unentbehrlich. Sollten Sie keine Treiber zur Ansteuerung Ihrer Hardware benötigen, können Sie den Befehl in der Startup-Sequence ersatzlos streichen. Hierdurch beschleunigt sich der Systemstart um ein paar Sekunden. Der Ordner Expansion ist in einem solchen Fall natürlich auch überflüssig und kann gelöscht werden.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.2.17 Mount

Syntax: `DEVICE/A`

Mit Hilfe des Mount-Befehls kann das AmigaDOS um neue Geräte erweitert werden. In der Grundkonfiguration kennt der Amiga die folgenden Geräte:

DF0: Das interne Diskettenlaufwerk.
PRT: Den Drucker.
PAR: Die parallele Schnittstelle.
SER: Die serielle Schnittstelle.
RAW: Die Fenster-Schnittstelle Raw.
CON: Die Fenster-Schnittstelle Con.
RAM: Die RAM-Disk.

Diese Geräte können nach einem Neustart des Rechners sofort angesprochen werden. Sollen weitere Geräte installiert werden (z.B. die Partitions einer Festplatte), so kann dies über den Mount-Befehl geschehen. Als Parameter erwartet Mount den Namen des neuen Gerätes. Unter diesem Namen müssen sich in der Text-Datei Mountlist, die sich im Verzeichnis devs der Systemdiskette befindet, Informationen zu dem Gerät befinden. Als Beispiel hierfür soll der Mount-List-Eintrag für ein 5¼-Zoll-Laufwerk dienen, das unter dem Namen DF2: angesprochen werden soll:

```
DF2:        Device = trackdisk.device
            Unit   = 2
            Flags  = 1
            Surfaces  = 2
            BlocksPerTrack = 11
            Reserved = 2
            PreAlloc = 11
            Interleave = 0
            LowCyl = 0
            HighCyl = 39
            Buffers = 5
            BufMemType = 3
#
```

In die Mount-List können beliebig viele Geräte eingetragen werden. Jede Eintragung muß jedoch mit dem Gerätenamen (hier: DF2:) beginnen und mit einem Lattenkreuz (#) enden. Dazwischen befinden sich die Angaben, die das Gerät näher

spezifizieren. Dabei werden die folgenden Schlüsselwörter von Mount ausgewertet:

Speziell bei Laufwerken:

Schlüsselwort	Funktion
Device	Name des Gerätetreibers.
Unit	Gerätenummer (z.B. 0, für df0:).
FileSystem	Bezeichnung eines speziellen FileSystems.
Priority	Priorität der Task (meist 10).
Flags	Parameter für OpenDevice (gewöhnlich 0).
Surfaces	Anz. Seiten des Laufwerks (bei Disketten: 2).
BlockPerTrack	Anzahl Blöcke pro Track.
Reserved	Anzahl Boot-Blöcke (gewöhnlich 2).
PreAlloc	(keine Funktion)
Interleave	Device-spezifisch (gewöhnlich 0).
LowCyl	Nummer des kleinsten Tracks.
HighCyl	Nummer des größten Tracks.
Buffers	Größe des Pufferspeichers in Blöcken.
BufMemType	Art des Speichers: 0,1 = CHIP- oder FAST-RAM 2,3 = Nur CHIP-RAM 4,5 = Nur FAST-RAM
Mount	Falls = 1, dann wird Devive sofort eingebunden. Falls = -1, dann wird Device beim ersten Zugriff eingebunden.

Bei anderen Geräten:

Schlüsselwort	Funktion
Handler	Pfadangabe zu dem Gerätetreiber.
Stack	Größe des Prozessor-Stacks für die Task.
Mount	Siehe oben.

Der Befehl unter Workbench 1.3

Syntax: `DEVICE/A,FROM/K`

Die Mount-List darf jetzt einen beliebigen Namen bekommen, der dem Mount-Befehl hinter dem Schlüsselwort FROM mitgeteilt werden muß:

```
mount df2: FROM devs:Geraeteliste_1
```

Fehlt die FROM-Angabe, sucht der Mount-Befehl weiterhin im Verzeichnis devs: nach der Datei Mountlist.

Dem Mount-Befehl kommt unter der Workbench-Version 1.3 eine besondere Bedeutung zu, da sich mit seiner Hilfe einige neue Geräte installieren lassen, die wir im folgenden kurz vorstellen möchten. Im Kapitel 4 werden sie dann ausführlich behandelt.

Aux

Eine serielle Schnittstelle, die im Gegensatz zu SER: die Daten nicht mehr zwischenspeichert. Die notwendigen Eintragungen in der Mount-List sind schon vorhanden, so daß die Schnittstelle durch *mount aux:* sofort installiert werden kann.

Pipe

Mit Hilfe dieses Gerätes können verschiedene Prozesse untereinander Daten austauschen. Möchte man beispielsweise von einem CLI aus eine Mitteilung an ein zweites CLI schicken, kann dies sehr leicht mittels der Pipe bewerkstelligt werden:

```
Eingabe im 1.CLI:    echo "Hallo CLI 2, wie geht's ?" to pipe:
```

Im zweiten CLI kann nun diese Information aus der Pipe ausgelesen werden:

```
Eingabe im 2. CLI:   type pipe:
Ausgabe:             Hallo CLI 2, wie geht's ?
```

Die Angaben zur Installation der Pipe sind in der Mount-List schon vorhanden.

CARD

Die CARD: stellt eine resetfeste RAM-Disk dar. Im Gegensatz du dem Device RAM: bleiben die Daten auch nach einem Reset des Rechners erhalten. Selbst eine Guru-Meditation kann der CARD in den meisten Fällen nichts anhaben. Leider erfolgt die Speicherverwaltung nicht dynamisch, so daß die CARD den ihr zugeteilten Systemspeicher auch dann voll in Anspruch nimmt,

falls sie keine Daten enthält. Mit den Angaben in der Mount-List kann man die Kapazität der CARD festlegen.

NewCon

Hiebei handelt es sich um eine neue Fensterschnittstelle, die eine Erweiterung des bekanntes Con-Fensters darstellt. Das NewCon-Device verwaltet einen 2 KByte großen Puffer, in dem jeweils die letzten Eingaben zwischengespeichert werden. Mit Hilfe der Cursor-Tasten kann man sich alte Eingaben zurückholen und nun auch Zeilen edieren. Das Con-Device eignet sich hervorragend in Verbindung mit dem NewCLI-Befehl.

Speak

Die Sprachausgabe des Amiga ist jetzt auch über das Gerät Speak ansprechbar. Dadurch kann man sich zum Beispiel das Inhaltsverzeichnis einer Diskette zur Abwechslung einmal vorlesen lassen: *dir >speak:*.

Der neue Mount-Befehl wertet zusätzlich zu den oben schon beschriebenen Schlüsselwörtern die folgenden Angaben aus:

`MaxTransfer`	**Max. Anzahl Blöcke, die übertragen werden.**
`Mask`	**Bestimmt Adreßbereich, den die DMA adressieren kann.**

Diese Angaben werden jedoch nur in Verbindung mit dem neuen Fast Filing System ausgewertet.

2.3 Befehle für Batch-Dateien

In diesem Abschnitt finden Sie Informationen zu den Befehlen, die speziell in Verbindung mit Batch-Dateien eingesetzt werden. Bei den Batch-Files, die oft auch Stapeldateien genannt werden, handelt es sich um reine Text-Dateien, die eine beliebige Anzahl aller CLI-Befehle enthalten und durch den Befehl Execute ausgeführt werden können. Mit diesem umfangreichen Themengebiet setzt sich das gesamte Kapitel 5 auseinander. Dort finden Sie auch eine ausführlichere Erklärung und jede Menge sinnvolle Anwendungen zu den Batch-Files.

2.3.1 Execute

Syntax: `execute NAME`

Da es sich bei den Batch-Files um reine Text-Dateien handelt, die mit nahezu jedem Editor oder Textverarbeitungsprogramm erstellt werden können (Beispielsweise mit dem Ed, der in 2.4 besprochen wird), stellen sie kein Programm dar, das durch die einfache Eingabe des File-Namens gestartet werden kann. Versucht man dies doch, antwortet der Rechner mit der Fehlermeldung 121: "file is not an object module", was diesen Sachverhalt auf eine kurze Formel bringt.

Um solche Stapeldateien ausführen zu lassen, benötigt man den Execute-Befehl. Er erwartet zumindest noch den File-Namen der Datei, die ausgeführt werden soll. Beispiel:

Die Batch-Datei Drucken enthalte die beiden folgenden Zeilen:

```
type Texte/Brief to prt:
date
```

Die Eingabe von *execute drucken* bewirkt nun dasselbe, als würde man vom CLI aus die beiden Zeilen einzeln eingeben: Zuerst wird der Brief auf dem Drucker ausgegeben, anschließend wird das aktuelle Datum und die Uhrzeit angezeigt.

Wie bei den meisten CLI-Befehlen kann man an den Execute-Befehl außer den File-Namen der auszuführenden Datei auch noch weitere Parameter hängen, die dann an das Batch-File übergeben werden. Die Stapeldatei muß in solchen Fällen eine entsprechende Anzahl Variablen bereitstellen, in die die Angaben übernommen werden.

Als Beispiel hierfür soll das obige Batch-File dienen. Anstelle der starren Vorgabe des auszudruckenden Textes (Texte/Brief) kann nun eine Variable eingeführt werden, an die jeder beliebige Name übergeben werden kann:

```
.key name
type <name> to prt:
date
```

Aufgerufen wird die Datei Drucken nun durch:

```
execute drucken Texte/Brief
```

Für die Benutzung solcher Variablen gibt es einige Regeln, die im folgenden besprochen werden.

1. Die Anweisung .key, mit der Variablen deklariert werden, muß sich immer zu Beginn eines Batch-Files befinden.

2. Soll die Übergabe mehrerer Parameter erlaubt sein, müssen die Namen der zugeordneten Variablen durch ein Komma voneinander getrennt werden. Es darf jedoch nur einmal die Anweisung .key benutzt werden, da sonst die Fehlermeldung "Execute: More than one K directive" auftritt. Beispiel für eine korrekte Anwendung:

```
.key dateiname,zielgerät
copy <dateiname> to <zielgerät>
```

3. Wie man aus den Beipielen schon entnehmen kann, wird eine in spitze Klammern (< bzw. >) gesetzte Variable durch ihren Inhalt ersetzt. An ihrer Stelle erscheint also nicht etwa der Variablenname, sondern die Angabe, die man ihr beim Execute-Befehl zugewiesen hat.

Normalerweise reicht die Beachtung dieser drei Punkte bei der Arbeit mit Variablen in Verbindung mit Batch-Files völlig aus. Es gibt jedoch noch einige zusätzliche Funktionen, die hier nicht verschwiegen werden sollen.

Zusätzlich zu dem Befehl .keys gibt es noch eine Anzahl weiterer Anweisungen, die in einem Batch-File durch einen Punkt eingeleitet werden können:

.def

Hiermit ist es möglich, einer Variablen einen bestimmten Inhalt zuzuweisen. Diese Anweisung darf überall im Text auftauchen. Eine Anwendung hierfür ist die Zuweisung eines festen Namens an eine Variable, für den Fall, daß dem Execute-Befehl ein

Name nicht ausdrücklich mitgegeben wurde (Default-Name). Ein solches Batch-File könnte folgendermaßen aussehen:

```
.key datei,gerätename
.def gerätename prt:
type <datei> to <gerätename>
```

Wird dem Execute-Befehl kein Gerätename hinzugefügt, so geht die Ausgabe der angegebenen Datei automatisch zum Drucker.

Speziell für diese Anwendung gibt es jedoch noch ein vereinfachtes Verfahren. Dazu muß der Variablenname in den spitzen Klammern um ein Dollarzeichen und den Text erweitert werden, der an die Stelle der Variablen treten soll, falls dem Execute-Befehl kein Parameter mitgegeben wird. Das obige Beipiel würde dann folgendermaßen aussehen:

```
.key datei,gerätename
type <datei> to <gerätename$prt:>
```

Falls zwar ein Dateiname, aber kein Gerätename angegeben wird, geht die Ausgabe automatisch auf den Drucker (prt:). Kommen wir nun zu den weiteren Anweisungen.

.dol

Hierdurch kann das Dollarzeichen zur Einleitung des Default-Textes (s.o.) durch ein beliebiges anderes Zeichen ersetzt werden. Zum Beispiel:

```
.dol #
```

Die entsprechende Zeile aus dem Beispiel unter .def müßte nun lauten:

```
type <datei> to <gerätename#prt:>
```

.bra

Besitzt eine ähnliche Aufgabe wie .dol. Hiermit kann jedoch für die spitze Klammer nach links (<) ein anderes Zeichen ausgewählt werden.

.ket

Wie .bra, jedoch diesmal für die rechte Klammer. Die beiden Bezeichnungen stellen zusammen eine Eselsbrücke dar (engl.: Bracket = Klammer).

.

Der Punkt, gefolgt von mindestens einem Leerzeichen, dient zur Kennzeichnung einer Kommentarzeile. BASIC-Programmierer benutzen hierfür die REM-Funktion.

.dot

Hiermit kann sogar der Punkt vor den Anweisungen gegen ein anderes Zeichen ausgetauscht werden.

Man sollte bei eigenen Batch-Dateien jedoch möglichst auf die Verwendung anderer Steuerzeichen verzichten, da diese nur zur Verwirrung beitragen.

Der Befehl unter Workbench 1.3

Auch weiterhin werden Batch-Dateien durch Execute aufgerufen. Durch die Einführung des Status-Flags s (Script-Flag) ist es jetzt jedoch auch möglich, eine Batch-Datei durch die alleinige Eingabe des File-Namens zu starten. Dazu muß jedoch vorher mit Hilfe des Protect-Befehls das Script-Flag gesetzt werden. Für ein Batch-File mit dem Namen Test_Batch sieht die Befehlsfolge zum Setzen des Flags zum Beispiel folgendermaßen aus:

```
protect Test_Batch +s
```

2.3.2 Echo

Syntax: `echo TEXT`

Zur Erholung nun ein einfacherer Befehl. Echo ermöglicht die Ausgabe einer Zeichenkette auf ein beliebiges Ausgabegerät. Im einfachsten Fall ist dies der Bildschirm:

```
echo "Hallo Otto !"
```

Möchte man die Ausgabe auf ein anderes Gerät umleiten, kann man dies (wie bei jedem anderen Befehl auch) durch die spitze Klammer erreichen.

```
echo >prt: "Hey, Du da: Kauf mir ein neues Farbband !"
```

Die Anführungsstriche sind übrigens nur dann notwendig, wenn der Text Leerzeichen enthält.

Als einzige Besonderheit gibt es bei dem Echo-Befehl die Zeichenkombination **n*, die an jeder Stelle des auszugebenden Textes stehen darf und einen Zeilenvorschub erzeugt. Beispiel:

```
echo "Vorsicht*n        Stufe!"
```

Ausgabe auf dem Bildschirm:

```
Vorsicht
        Stufe!
```

Sollte einmal der seltene Fall eintreten, daß man die Zeichenkombination **n* als echten Teil des auszugebenden Textes benötigt, ist hierfür die Zeichenkette ***n* einzusetzen.

Der Befehl unter Workbench 1.3

Syntax: `echo ,NOLINE/S`

Neu hinzugekommen ist die Option NOLINE, die den normalerweise erfolgenden Zeilenvorschub nach einer Ausgabe mit Echo unterbindet.

2.3.3 Failat

Syntax: `failat RClim`

Jeder CLI-Befehl und viele andere Programme auch geben eine Fehlernummer zurück, falls bei ihrer Ausführung irgendeine Besonderheit aufgetreten ist. Im CLI ist den meisten Nummern ein bestimmter Fehlertext zugeordnet, den man sich mit *Fault*

gefolgt von der jeweiligen Nummer auch *manuell* ausgeben lassen kann. Die Nummer 216 zum Beispiel bedeutet, daß soeben versucht worden ist, einen Ordner zu löschen, der noch Einträge enthält.

Tritt nun während der Abarbeitung von CLI-Befehlen innerhalb eines Batch-Files eine Fehlernummer auf, die größer oder gleich 10 ist, wird die Abarbeitung der Stapeldatei sofort unterbrochen und zum übergeordneten Programm (z.B. zum CLI) zurückgekehrt. Diese Fehlergrenze kann jederzeit durch den Befehl Failat erfragt werden. Das ist natürlich nur sinnvoll, da man die Grenze beliebig festlegen kann. In einigen Fällen ist es z.B. wünschenswert, wenn ein Batch-File schon beim Auftreten einer Warnung (also einer Fehlernummer kleiner als 10) abbricht.

Hierzu eine Anwendung: Bei Compilern wird meist zwischen Warnungen und echten Fehlern unterschieden. Warnungen können in vielen Fällen zwar ignoriert werden, zeugen jedoch von einem schlechten Programmierstil. Setzt man die Fehlergrenze in einem Batch-File eines Compilers auf z.B. 1 herab, wird die Arbeit bei jeder auftretenden Unregelmäßigkeit abgebrochen. Hierdurch wird verhindert, daß eventuell noch weitere Arbeitsgänge (Assemblieren, Linken) aufgerufen werden.

Um in einem Batch-File die Fehlergrenze neu zu setzen, muß Failat einfach um die neue Fehlernummer ergänzt werden, bei der ein Abbruch frühestens stattfinden soll (RClim = Return Condition Limit, vgl. Syntax-Liste). Diese neue Grenze gilt jedoch nur für die Zeit, in der die Stapeldatei abgearbeitet wird. Sie wird anschließend automatisch wieder auf 10 zurückgesetzt.

Wird direkt vom CLI aus eine neue Fehlergrenze vorgegeben, gilt diese auch in einer von diesem CLI aus aufgerufenen Stapeldatei. Wird sie hier wieder umdefiniert, liegt die Fehlergrenze nach der Rückkehr zum CLI in jedem Fall wieder bei 10. Taucht der Failat-Befehl in der Datei jedoch gar nicht auf oder dient er nur zur Abfrage, ändert sich die im CLI eingegebene Fehlergrenze nicht.

Jedes neu aufgerufene CLI übernimmt automatisch auch die aktuelle Fehlergrenze des aufrufenden CLI. Nach dem Aufruf können die Grenzen aber unabhängig voneinander verändert werden.

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.3.4 Quit

Syntax: `quit RC`

Der Quit-Befehl ermöglicht das Verlassen eines Batch-Files von einer beliebigen Stelle aus. Am Ende einer Stapeldatei ist Quit jedoch nicht erforderlich. Möchte man dem Programm, zu dem zurückgekehrt wird, das Auftreten einer Unregelmäßigkeit mitteilen, kann Quit noch um die gewünschte Fehlernummer (RC = Return Code) erweitert werden. Kehrt die Kontrolle an das CLI zurück, so erscheint bei einer Fehlernummer größer oder gleich 10 der Text:

```
quit failed returncode xx
```

(xx steht hierbei für die dem Quit-Befehl mitgegebene Fehlernummer.)

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.3.5 If/Else/EndIf

Diesmal werden gleich drei Befehle auf einmal behandelt, und zwar nicht, damit wir schneller fertig werden, sondern weil die Kommandos Else bzw. EndIf nur in Verbindung mit If auftreten dürfen.

Wahrscheinlich können Sie sich schon denken, wozu die Befehle benötigt werden: Mit ihnen ist es möglich, bestimmte Teile eines Batch-Files nur unter ganz bestimmten Bedingungen abarbeiten zu lassen. Im einfachsten Fall benötigt man dazu nur die Befehle If und EndIf:

```
if exists Texte/Brief
type Texte/Brief to prt:
endif
echo "Hab ich den Brief jetzt gedruckt oder nicht ?"
...
...
```

In diesem Beispiel wird der Befehl Type nur dann abgearbeitet, falls eine Datei Brief im Unterverzeichnis Texte auch wirklich existiert. Falls nicht, wird die Abarbeitung des Batch-Files direkt hinter dem EndIf-Befehl fortgesetzt.

Außer der Bedingung Exists, mit der das Vorhandensein bestimmter Dateien auf einem Laufwerk oder der RAM-Disk abgefragt werden kann, gibt es noch eine Reihe weiterer Codes, die hinter If erscheinen dürfen:

EQ (engl.: equal = gleich)

Mit EQ kann abgefragt werden, ob zwei Texte den gleichen Inhalt besitzen. Zum Beispiel:

```
if "Das ist der Text" eq "Das ist der Text"
echo "Ja, beide Texte sind gleich !"
endif
```

So etwas bringt natürlich gar nichts, da man die Abfrage auch gleich weglassen kann. Von alleine wird sich der Text schon nicht verändern. Interessanter wird die Anwendung von EQ in Verbindung mit Batch-Variablen, die ja schon bei dem Befehl Execute behandelt worden sind. Dazu zwei Beispiele:

```
.key eingabe
if <eingabe> eq brief
echo "Sie haben das Wort 'brief' eingegeben !"
endif
```

oder:

```
.key eingabe
if <eingabe> eq ""
echo "Sie haben ja gar nichts eingegeben !!!"
endif
```

Hier muß man deutlich zwischen der Variablen (*eingabe*), dem Inhalt der Variablen (*<eingabe>*) und dem Text (*brief*) bzw. ("") unterscheiden.

Wichtig ist noch, daß bei Textvergleichen nicht zwischen Groß- und Kleinschrift unterschieden wird, *IF Brief EQ BRIEF* liefert also ein wahres Ergebnis.

FAIL

Durch *IF FAIL* kann abgefragt werden, ob der zuletzt bearbeitete Befehl eine Fehlernummer >=20 geliefert hat. Diese Auswertung kann natürlich nur dann stattfinden, wenn vor der Ausführung des Befehls, bei dem der Fehler aufgetreten ist, die Fehlergrenze von 10 auf einen größeren Wert als 20 gesetzt worden ist. Ansonsten würde die Abarbeitung des Batch-Files ja sofort unterbrochen.

ERROR

IF ERROR funktioniert ähnlich wie *IF FAIL*. Nur die Fehlergrenze liegt in diesem Fall schon bei 10.

WARN

Die Fehlergrenze bei *IF WARN* liegt bei 5. Daher ist es hier nicht notwendig, mit Failat die Standardgrenze von 10 auf einen höheren Wert zu setzen.

Man sollte sich aber von den Bezeichnungen WARN und ERROR nicht irreführen lassen: Wenn eine auftretende Fehlernummer 225 zum Beispiel durch *IF WARN* abgefangen wird, ist dies selbstverständlich keine Warnung, sondern ein dicker FAIL-ERROR. Es empfiehlt sich, nach einem Hochsetzen der Fehlergrenze mit Failat zuerst die dicken Fische abzufangen.

NOT

Wird der Zusatz NOT vor eine der Bedingungen unter 1. bis 4. gesetzt, dreht sich der Wahrheitsgehalt der Aussage genau um. Dazu ein etwas umfangreicheres Beipiel:

```
.key text
if not exists <text>
echo "So eine Datei hab' ich nicht !"
endif
if exists <text>
echo "Los geht's !"
type <text> to prt:
endif
```

Das Batch-File benötigt beim Aufruf noch den Namen einer beliebigen Textdatei, der in der Variablen Text abgelegt wird. Im ersten Teil wird nun geprüft, ob die angegebene Datei vielleicht gar nicht existiert. In einem solchen Fall erscheint der erste Echo-Text.

Anschließend wird noch einmal mit *IF EXISTS* getestet, diesmal aber, ob die Datei vorhanden ist. Nur dann wird sie auf dem Drucker ausgegeben. (Funktioniert nur bei echten Text-Files!).

Dieses Beispiel dient dann auch gleich als idealer Übergang zu der Erklärung des Else-Befehls. Mit Else kann nämlich sehr leicht eine Alternative für eine negativ verlaufende If-Abfrage in ein Batch-File eingebaut werden. Das sieht, auf das obige Problem übertragen, folgendermaßen aus:

```
.key text
if not exists <text>
 echo "So eine Datei hab' ich nicht !"
else
 echo "Los geht's !"
 type <text> to prt:
endif
```

Dieses Batch-File liefert dasselbe Ergebnis wie das vorherige, jedoch mit weniger Aufwand und auch schneller, da ein Steuerbefehl weniger geladen werden muß.

Zuletzt noch einige allgemeine Anmerkungen zu den drei Befehlen: Jeder Anweisungsblock darf eine beliebige Anzahl Zeilen

besitzen. Er muß entweder durch ein Else oder ein EndIf abgeschlossen werden. Innerhalb eines solchen Blocks dürfen sogar beliebig viele If-Befehle verschachtelt werden. Ein Else oder EndIf bezieht sich hierbei immer auf das zuletzt ausgeführte If.

Zur vollständigen Verwirrung folgt ein letztes Beispiel. Es wertet aus, wie viele Parameter dem Execute-Befehl mitgegeben worden sind (max. drei). Durch die strukturierte Darstellung wird es Ihnen sicher leichter fallen, die Funktionsweise des Programms zu durchschauen.

```
.key text1,text2,text3
if not <text1> eq ""
   if not <text2> eq ""
      if not <text3> eq ""
         echo "Alle drei Angaben sind vorhanden !"
      else
         echo "Die dritte Angabe fehlt !"
      endif
   else
      echo "Die zweite und dritte Angabe fehlen !"
   endif
else
   echo "Sie Schlumpf haben gar keine Angaben gemacht !"
endif
```

Die Befehle unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.3.6 Ask

Syntax: `ask PROMPT/A`

Es ist unverständlich, warum dieser schöne Befehl in kaum einer Publikation erwähnt wird. Weiß vielleicht niemand, wie man ihn einsetzt? Dabei ist doch nichts einfacher als das:

Der Ask-Befehl kann entweder alleine stehen oder noch um einen Text ergänzt werden, der bei Aufruf der Funktion erscheinen soll. Nun wartet der Rechner auf eine Antwort, die entweder *Yes* oder *No* (bzw. *Y* oder *N*) gefolgt von Return sein darf. Wird etwas anderes eingegeben, bleibt der Ask-Befehl

hartnäckig und fragt solange, bis ihm die Antwort gefällt. Die Auswertung erfolgt nun über den Fehlercode Nummer 5, den der Befehl bei einer bestätigenden Eingabe (und auch nur dann!) zurückliefert. Eine erste Reaktion auf unterschiedliche Eingaben demonstriert das folgende Beispiel:

```
failat 5
ask "Soll ich sofort abbrechen ? (y/n)"
echo "Na gut, dann mach ich halt noch weiter !"
...
```

Da normalerweise die Fehlergrenze, bei der die Abarbeitung eines Batch-Files abgebrochen wird, bei 10 liegt, muß sie zunächst auf 5 herabgesetzt werden, damit bei Eingabe von *Yes* auch tatsächlich zum CLI zurückgesprungen wird.

Diese Lösung wird jedoch kaum jemanden richtig zufriedenstellen. Besser wäre es, wenn man auf die Reaktion des Anwenders hin zwei völlig unterschiedliche Programmteile abarbeiten lassen könnte. Was liegt da näher, als hierfür das *IF WARN* aus dem Abschnitt 2.3.5 zu benutzen. Damit entfällt auch das lästige Heruntersetzen der Fehlergrenze auf einen Wert kleiner als Sechs. Beispiel:

```
ask "Kennen Sie nun den Ask-Befehl ? (y/n)"
if warn
   echo "Sehr gut, weiter so !"
else
   echo "Sechs, setzen !"
endif
```

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.3.7 Skip/Lab

Taucht bei der Bearbeitung eines Batch-Files an einer Stelle der Befehl Skip auf, so wird der Dateitext auf das nächste Auftreten des Wörtchens Lab (engl.: Label = Etikett, Kennzeichnung) durchsucht und die Abarbeitung an dieser Stelle fortgesetzt.

Wird der Skip-Befehl jedoch noch um einen Namen erweitert, wird zu einem gleichnamigen Label gesprungen. Als Beispiel:

```
ask "Können Sie mit Skip und Lab umgehen ? (y/n)"
if warn
skip marke
endif
echo "Auch nicht so schlimm !"
quit
lab marke
echo "Bloß nicht zu oft verwenden !
```

Der Befehl Skip weist also gewisse Ähnlickeiten mit dem GOTO (für BASIC- oder C-Programmierer) oder dem JMP (für Assembler-Experten) auf. Einen entscheidenden Unterschied gibt es jedoch: Mit Skip kann man im Programmtext nicht zurückspringen! Wie in jeder anderen Programmiersprache, die eine Sprungfunktion bietet, sollte man auch bei den Batch-Files etwas zurückhaltend damit umgehen, da der Befehl nicht gerade zur Übersichtlichkeit des Programmtextes beiträgt. Im obigen Beispiel ist sogar äußert unsauber programmiert worden, da hier aus einer If/EndIf-Konstruktion herausgesprungen wird. Das läßt sich beim Skip aber meistens gar nicht verhindern. In nahezu allen Fällen stellt eine reine If/Else/EndIf-Konstruktion die bessere Lösung dar.

Die Befehle unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.3.8 Wait

Meistenteils kann ein Rechner gar nicht schnell genug sein. In einigen Fällen ist die Schnelligkeit aber eher hinderlich. Hierfür existiert der Befehl Wait, der eine vorgegebene Zeit lang gar nichts macht.

Ein typisches Beispiel für unbrauchbare Schnelligkeit ist die Ausführung zweier Prozesse, die gleichzeitig auf ein und dasselbe Laufwerk (RAM-Disk ausgenommen) zugreifen. Listet man sich in einem CLI-Fenster das Directory der Diskette in Laufwerk 0 durch *dir* auf und gibt von einem zweiten CLI aus

den Befehl *list df0:* ein, werden beide Befehle parallel bearbeitet. Im Endeffekt dauert das aber länger, als wenn man die Befehle nacheinander eingeben würde. Da sich beide Prozesse die Floppy teilen müssen, kann sich jeder Befehl während seiner Rechenzeit nur einige Tracks von der Diskette holen. Sehr viel Zeit wird nun dadurch verschenkt, daß der Floppykopf dauernd seine Position ändern muß.

Noch extremer fällt dieser Effekt auf, wenn man z.B. in der Startup-Sequence gleichzeitig zwei Programme lädt, da hierbei meist die Daten nicht ganz so verstreut auf der Diskette liegen wie bei dem Directory. Liegen die Daten von Programm Nr. 1 in der Nähe von Spur Null, und die Daten von Programm Nr. 2 in der Nähe von Spur 79, könnte man meinen, die Diskette übersteht die Prozedur nicht, da der Kopf andauernd die knapp 80 Tracks überspringen muß.

Hier empfiehlt es sich, erst die Ladezeit des ersten Programms abzuwarten. Dies kann sehr einfach durch den Befehl Wait geschehen, der noch die abzuwartende Zeit als Parameter erwartet. Die Zeit kann entweder in Sekunden oder Minuten angegeben werden. Es kann aber auch eine Systemuhrzeit vorgegeben werden, ab der die Arbeit fortgesetzt werden soll. Wait ohne Parameter wartet eine Sekunde lang. Hierzu wieder einige Beipiele:

```
1. wait                 Wartet 1 Sekunde.
2. wait 5               Wartet 5 Sekunden.
3. wait 5 sec           Wie 2.
4. wait min             Wartet 1 Minute.
5. wait 5 min           Wartet 5 Minuten.
6. wait until 14:30     Wartet bis 14:30 Uhr.
```

Natürlich kann auch die Funktion Wait jederzeit durch die Tastenkombination Ctrl und C abgebrochen werden.

Wie man mit dem Wait-Befehl auch schnell einen kleinen CLI-Wecker erzeugen kann, wird im Kapitel 4 gezeigt.

2.3.9 Version

Syntax: `version [<library name>] [version] [revision]`

Mit Hilfe der Funktion Version kann man die derzeitige Versions- und Revisionsnummer der Workbench, eines Gerätes oder einer Library herausfinden. Gibt man den Befehl Version ohne Parameter ein, so erhält man Angaben über die eigene Kickstart- und Workbench-Version, z.B.:

```
Kickstart version 33.180. Workbench version 34.4
```

Ferner kann Version auch durch einen speziellen Geräte- oder Library-Namen ergänzt werden:

```
Eingabe:    version trackdisk.device
Ausgabe:    trackdisk.device version 33.127
```

Schließlich besteht noch die Möglichkeit, einen Test auf eine bestimmte Versionsnummer vorzunehmen. Ist die angegebene Versionsnummer größer als die getestete, so wird der Errorcode 5 zurückgeliefert. Der Fehlerstatus kann innerhalb einer Batch-Datei mit Hilfe einer If/Else-Konstruktion ausgewertet werden. Die folgende Batch-Datei ruft das Programm Mathe nur dann auf, falls der Amiga mit der schnelleren Mathe-Bibliothek der Version 34.44 oder jünger ausgestattet ist. Andernfalls wird eine entsprechende Meldung ausgegeben.

```
version >nil: mathieeedoubbas.library 33 44
if warn
   echo "Die schnelle Mathe-Bibliothek ist nicht da !"
else
   run Mathe
endif
```

Der Befehl unter Workbench 1.3

Es haben sich keine Änderungen ergeben.

2.4 Die Editoren ED und Edit

Die beiden Programme ED und Edit sind schwer in die schon vorhandenen drei CLI-Themengebiete einzuordnen, darum wird ihnen hier ein eigener vierter Abschnitt gewidmet. Es handelt sich bei ihnen um zwei etwas umfangreichere Programme (jeweils ca. 19 KByte), die eine Bearbeitung von reinen Textdateien ermöglichen.

Den ED könnte man als einen Full-Window-Editor bezeichnen. Es handelt sich also um einen Editor, der den gesamten Text in den Arbeitsspeicher des Amiga lädt und davon je nach gewählter Fenstergröße einen entsprechend großen Ausschnitt anzeigt. Mit Hilfe der vier Pfeiltasten kann man die Schreibposition (den Cursor) an eine beliebige Stelle innerhalb des Fensters setzen und hier Änderungen am Text vornehmen. Versucht man dabei, über den oberen oder unteren Fensterrand hinauszugehen, wird sofort der Textausschnitt entsprechend zeilenweise verschoben (gescrollt). Wird ein Text bearbeitet, der mehr Spalten besitzt als das ED-Fenster anzeigen kann, wird bei Randkontakt des Cursors automatisch auf die angrenzende Texthälfte umgeschaltet. Wer schon mit AmigaBASIC gearbeitet hat, wird zumindest das Prinzip (nicht die Schnelligkeit!) kennen. Der große Konkurrent des ED hat den etwas längeren Namen Edit bekommen. Er gehört zur Kategorie der Line-by-Line-Editoren. Der grundsätzliche Unterschied zum ED besteht darin, daß die Bearbeitung einer Textdatei hier zeilenweise durch die Eingabe verschiedener Befehle erfolgt. Änderungen können also hier nicht einfach durch direkte Manipulation innerhalb eines Fensters vorgenommen werden. Ein bekannter Vertreter dieser Editorart ist das CLI, in dem wie beim Edit immer nur die letzte Zeile Befehle aufnehmen kann.

Die Anwendungsgebiete für die beiden Editoren sind vielfältig. Meistens werden sie zur Bearbeitung von Batch-Files oder zur Erstellung des Quell-Codes bei der Programmierung in Compilersprachen eingesetzt.

In den meisten Fällen wird man dabei auf den ED zurückgreifen, da dieses Programm viel einfacher zu bedienen ist und

man vor allem eine bessere Übersicht über den Text hat, was ja gerade beim Programmieren besonders wichtig ist.

Die Vorteile des Edit liegen hauptsächlich in dem relativ geringen Bedarf an Speicher, da die Texte nicht vollständig geladen werden müssen. Bei der Bearbeitung sehr großer Dateien, bei denen der ED einfach nicht mehr mitspielt, kann nur noch Edit helfen. Er ermöglicht sogar das gleichzeitige Öffnen mehrerer Quelltextdateien, aus denen wahlweise Textpassagen bearbeitet und in eine Zieldatei übernommen werden können. Insgesamt stellt Edit somit sicher das flexiblere Programm dar. Über die Möglichkeiten, die Edit bietet, könnte ein eigenes Buch geschrieben werden, weshalb hier nur auf die grundlegenden Funktionen eingegangen werden kann.

2.4.1 ED

Zunächst wird der wohl am häufigsten eingesetzte ED behandelt. Er wird im Normalfall aufgerufen durch:

```
ed texte/prog
```

Der Zusatz Texte/Prog stellt hierbei den Pfadnamen auf eine zu edierende Textdatei dar. Hierbei können zwei verschiedene Fälle auftreten:

1. Die Datei existiert schon in dem angegebenen Verzeichnis.

In diesem Fall wird versucht, die Datei in den Arbeitsspeicher zu laden. Gelingt dies, so erscheint anschließend das Fenster des ED, in dem je nach Format und Größe der Datei entweder der gesamte Text oder nur ein Ausschnitt hiervon dargestellt wird. Wird der Ladevorgang jedoch vorzeitig abgebrochen, kann dies an mangelndem Speicherplatz liegen ("Unable to open window ...") oder daran, daß die angegebene Datei nicht edierbar ist, da es sich entweder um einen Ordner handelt oder sie nicht darstellbare Zeichen enthält. Man erkennt dies an den Meldungen: "x is a directory and cannot be edited" - - "File contains binary."

2. Eine Datei mit diesem Namen gibt es in dem angegebenen Ordner nicht.

Dabei erscheint nur das leere Fenster des ED, vorausgesetzt, es ist genügend Systemspeicher vorhanden. Es wird jedoch noch nicht eine gleichnamige Datei auf der Diskette erzeugt.

Beim Start des ED kann zusätzlich zu dem File-Namen noch die Größe des gewünschten Arbeitsspeichers in Bytes angegeben werden. Als Beipiel:

```
ed texte/prog size 60000
```

Fehlt diese Angabe, reserviert der ED automatisch 40000 Bytes, es sei denn, es wird eine größere Datei geladen.

Nun kann die Arbeit mit dem ED auch schon beginnen. Nachdem man die Größe und die Position des Fensters festgelegt hat, kann man mit den Cursor-Tasten beliebig "umherflitzen" und vorhandenen Text manipulieren bzw. neuen Text einfügen. Die Backspace- und die Del-Taste funktionieren, wie man es gewöhnt ist: Backspace löscht das Zeichen links vom Cursor, Del entfernt das Zeichen, auf dem der Cursor gerade steht. Die Maus wird vom ED leider nicht beachtet.

Es gibt zwei verschiedene Möglichkeiten, dem ED Befehle mitzuteilen:

1. Direkt, durch gleichzeitiges Drücken der Ctrl-Taste und einem bestimmten Zeichen. Der zugehörige Befehl wird sofort ausgeführt, und man kann weiter den Text bearbeiten.

2. über die Befehlszeile, die am unteren Fensterrand auftaucht, sobald man die ESC-Taste betätigt. Man erkennt diese Zeile an dem * in der ersten Spalte. Solange das Sternchen angezeigt wird, befindet man sich im Befehlsmodus, und es können keine Veränderungen am Text vorgenommen werden. Erst nach der Eingabe und Ausführung eines Befehls oder durch ein einfaches Return wird dieser Modus wieder verlassen. Sollen mehrere Befehle hintereinander ausgeführt werden, kann man sie auch, durch ein Semikolon voneinander getrennt, in die gleiche Befehlszeile schreiben.

Ein Auflistung sämtlicher Befehle, die der ED ausführen kann, befindet sich am Schluß dieses Buches. An dieser Stelle sollen nur die wichtigsten Befehle aufgeführt werden, die aber in den meisten Fällen völlig ausreichen.

Befehle, die direkt ausgeführt werden

Ctrl und A

Es wird unter der aktuellen Zeile eine neue Zeile eingefügt.

Ctrl und B

Die aktuelle Zeile wird vollständig entfernt.

Ctrl und G

Wiederholt den zuletzt eingegebenen ESC-Befehl (Wichtig beim Suchen und Ersetzen!).

Ctrl und Y

Die Zeile wird ab der Cursor-Position bis zum Ende gelöscht.

Befehle, die nach ESC eingegeben werden müssen und erst durch Return ausgeführt werden (nach Wichtigkeit geordnet)

X (eXit)

Der Text wird zurückgeschrieben und der ED anschließend verlassen. Im Unterverzeichnis T der Diskette wird jedoch noch eine Kopie des unveränderten Textes mit dem Namen ed-backup abgelegt.

Q (Quit)

Die Arbeit mit dem ED wird beendet, ohne daß irgend etwas abgespeichert wird. Vorher muß der Befehl jedoch noch einmal bestätigt werden.

SA (SaveAs)

Hiermit kann man den Text abspeichern, ohne den ED anschließend zu verlassen. Soll der Text nicht unter dem Namen abgelegt werden, der dem ED beim Start mitgeteilt worden ist, kann hier auch in Anführungszeichen der Name einer neuen oder schon bestehenden Datei angegeben werden, die den Text aufnehmen soll.

Achtung: Der alte Inhalt einer schon bestehenden Datei ist danach unwiederbringlich verloren!

Es ist sogar möglich, die Daten direkt an ein Peripheriegerät zu schicken: *SA "prt:"* gibt z.B. den Text auf den Drucker aus.

J (Join)

Zieht zwei Zeilen zu einer einzigen zusammen. Sehr nützlich, wenn man aus Versehen eine Zeile durch Return getrennt hat. Der Cursor muß sich dabei am Ende des Textes der oberen Zeile befinden.

BS (BlockStart)

Dient zur Markierung eines Blockanfangs für verschiedene Blockoperationen. Die Zeile, in der sich der Cursor zu diesem Zeitpunkt befindet, wird als die oberste Zeile des Blockes angesehen.

BE (BlockEnd)

Macht die derzeitige Zeile, in der der Cursor steht, zur untersten Zeile des Blockes.

IB (InsertBlock)

Fügt eine Kopie des durch BS und BE markierten Blockes unter der augenblicklichen Cursor-Position ein.

DB (DeleteBlock)

Löscht den durch BS und BE markierten Block vollständig. Die Zeilen werden aus dem Text entfernt.

2.4.2 Edit

Sie sollten jetzt möglichst alles vergessen, was über die Bedienung des ED gesagt wurde, denn der Edit basiert auf einem völlig anderen Prinzip:

Im Arbeitsspeicher (Puffer) des Edit befindet sich immer nur eine begrenzte Anzahl Textzeilen, die im Normalfall vom Anwender nacheinander bearbeitet werden. Hat man die letzte Zeile dieses Puffers ediert, wird automatisch eine neue Zeile aus der Datei nachgeladen und die "vorderste" Zeile in die Zieldatei geschrieben. Es wird also grundsätzlich mit einer Quelldatei und einer Zieldatei gearbeitet. Dies ist ein wesentlicher Unterschied gegenüber dem ED!

Der Anwender hat nur die Möglichkeit, die Zeilen zu bearbeiten, die sich gerade im Puffer befinden. Es ist also möglich, eine beschränkte Anzahl Zeilen zurückzugehen. Hat eine Zeile jedoch den Puffer schon verlassen und ist somit in die Zieldatei geschrieben worden, ist sie für einen Ediervorgang nicht mehr zugänglich.

Wird die Arbeit mit Edit beendet, wird noch der derzeitige Pufferinhalt vollständig in die Zieldatei geschrieben. Außerdem muß dies auch mit dem eventuell noch vorhandenen Rest der Quelldatei geschehen, damit keine Daten verlorengehen. Wie später noch gezeigt wird, gibt es eine Möglichkeit, Edit zu verlassen, ohne daß noch Lese- oder Schreiboperationen stattfinden.

Edit bietet ferner die Möglichkeit, während der Arbeit verschiedene andere Quelldateien zu öffnen und auszulesen. Jede neue Datei wird dabei wie die ursprüngliche Quelldatei von Anfang an und zeilenweise abgearbeitet. Kehrt man zu einer schon einmal bearbeiteten, offenen Quelldatei zurück, wird automatisch an der alten Position weitergearbeitet.

Schließlich wartet Edit noch mit einer ganz besonderen Spezialität auf: Er kann seine Edier-Befehle nicht nur von der Tastatur lesen, sondern sogar von beliebigen Dateien, die man extra für diesen Zweck anlegen kann. Später werden hierzu noch einige Beispiele gezeigt.

Parameter des Edit

Daß man es bei Edit mit einem sehr vielseitigen Programm zu tun hat, bemerkt man schon an den vielen Parametern, die dem Editor beim Aufruf mitgegeben werden können. Die Syntax des Befehles lautet:

```
edit FROM/A,TO,WITH/K,VER/K,OPT/K
```

Die fünf Parameter haben die folgende Bedeutung:

FROM

Hier muß (/A) der File-Name der Datei angegeben werden, die bearbeitet werden soll. Anders als beim ED muß diese Datei schon existieren. Man kann also mit Edit keine vollständig neuen Texte erzeugen.

TO

Hier kann der Name einer Zieldatei angegeben werden, die den bearbeiteten Text aufnehmen soll. Fehlt dieser Name, geht Edit folgendermaßen vor:

Als Zieldatei wird im Unterverzeichnis t des aktuellen Laufwerkes eine Arbeitsdatei eingerichtet, die den edierten Text aufnimmt. Wird die Arbeit mit Edit beendet, bekommt diese Arbeitsdatei den vollständigen Pfadnamen der Quelldatei, die unter *From* angegeben wurde. Die ursprüngliche Quelldatei wird im Ordner T unter dem Namen Edit-Backup weitergeführt, bis sie bei einer erneuten Arbeit mit Edit wieder überschrieben wird.

WITH

Wie oben schon erwähnt wurde, kann Edit Befehle statt von der Tastatur auch aus bestimmten Befehlsdateien holen. In diesem Fall muß hinter WITH der Name einer solchen Datei stehen.

VER

Sämtliche Ausgaben des Edit, die normalerweise auf den Bildschirm gelangen, können durch VER auf ein beliebiges Gerät umgeleitet werden. Beispielsweise wird durch *Ver Datei* die Ausgabe auf ein File mit dem Namen Datei gelenkt. Mit *VER con:10/10/300/100/VER-Fenster* kann man sich in einem Fenster ansehen, was alles in solch eine Datei geschrieben wird.

OPT P

Hiermit kann die Anzahl der Zeilen, die sich gleichzeitig im Puffer befinden können, vorgewählt werden. Beispiel: OPT P100. Anstelle der normalen 40 Zeilen kann der Puffer nun 100 Zeilen aufnehmen. Das ist zwar praktischer, verbraucht aber auch mehr Systemspeicher.

OPT W

Ermöglicht es, die maximale Zeilenlänge auf einen anderen Wert als 120 zu setzen. Beipiel: OPT W81. Nun dürfen die Zeilen nur noch maximal 81 Zeichen lang sein.

OPT PxWy

Kombination von OPT P und OPT W. Für (x) bzw. (y) sind natürlich die gewünschten Werte einzusetzen.

Start des Edit

In den meisten Fällen kommt man beim Start des Edit mit der Angabe des Namens der zu bearbeitenden Datei aus. Wie oben schon erwähnt wurde, legt Edit in einem solchen Fall die edierten Zeilen zuerst in einer Hilfsdatei ab.

Nach dem Start erscheint das Edit-Prompt, das nur aus einem Doppelpunkt besteht. Es wartet, ähnlich wie im CLI, auf die Kommandos des Benutzers. Wegen der Zeilenorientiertheit muß man sich nun zunächst einmal die Zeile aussuchen, die bearbeitet werden soll. Um dies zu vereinfachen, numeriert Edit automatisch alle Zeilen des Quelltextes intern durch. Direkt nach dem Start ist die Zeile Nummer 1 des Quelltextes die aktuelle Zeile. Leider wird ihr Inhalt nicht automatisch angezeigt. Um zu einer anderen Zeile zu gelangen, gibt es nun unterschiedliche Methoden:

a. Durch die Eingabe von N (Next) gelangt man eine Zeile weiter.

b. Durch die Eingabe von P (Previous) gelangt man eine Zeile zurück.

c. Durch die Eingabe von Mx (Move) gelangt man zur Zeile mit der Nummer x.

Um zum Beispiel den Inhalt der Zeile 1 anzeigen zu lassen, genügt die Eingabe von N gefolgt von P. Man kann auch mehrere Befehle gleichzeitig eingeben, die dann aber wie beim ED durch ein Semikolon voneinander getrennt werden müssen. Zwischen Groß- und Kleinschreibung der Befehle unterscheidet auch Edit nicht.

Mit P bzw. M kann man jedoch nur bis zu der Zeile des Puffers zurückgehen, die noch nicht in die Zieldatei geschrieben worden ist. Sobald man eine Zeile mit einer Nummer größer als 40 anwählt, werden entsprechend viele Zeilen in die Zieldatei übernommen. Versucht man nun, z.B. wieder auf die Zeile mit der Nummer 1 zurückzugehen, erhält man die Fehlermeldung: "Line number 1 to small".

Vor die Befehle P und N kann zusätzlich noch eine Zahl gestellt werden, die angibt, wie oft das Kommando ausgeführt werden soll. Um zum Beispiel im Text 10 Zeilen weiterzugehen, genügt die Eingabe 10N.

Eine weitere Möglichkeit, eine bestimmte Zeile anzusteuern, stellt Edit mit dem Befehl F (Find) zur Verfügung. Mit ihm

kann man nach bestimmten Texten innerhalb der Datei suchen. Dem Befehl muß ein in beliebige Zeichen eingeschlossener Suchtext folgen. Zum Beispiel:

```
f ?Schlüssel?
```

Es wird ab der aktuellen Zeilennummer (einschließlich!) nach dem Auftreten des Wortes Schlüssel gesucht.

Fehlt die Angabe hinter F, geht der Befehl von dem zuletzt eingegebenen Suchtext aus. Dies ist sehr praktisch, wenn man nach mehreren Textstellen in der Datei sucht, die einen bestimmten Suchtext enthalten. Damit hierbei nicht immer die gleiche Zeile ausgegeben wird, muß nach einem geglückten Suchvorgang die nächste Zeile angewählt werden (zum Beispiel mit N).

Texte ändern

Nachdem man sich die entsprechende Zeile ausgesucht hat, kann es nun endlich mit den Änderungen losgehen. Selbst diese können jetzt nicht direkt (wie beim ED) in der Zeile vorgenommen werden, sondern müssen durch bestimmte Befehle veranlaßt werden. Die wichtigsten Kommandos, die Edit hierfür bereitstellt, sind:

e (exchange)

Ermöglicht das Ersetzen einer bestimmten Zeichenkette durch eine vorgegebene neue. Ein Beispiel: Vorher lautet die Zeile: "Der Edit ist schwer zu bedienen."

```
Eingabe: e/schwer/leicht/ Nun steht dort: Der Edit ist leicht zu
bedienen.
```

oder:

```
Eingabe: e/schwer // Nun steht dort: Der Edit ist zu bedienen.
```

a (after)

Es wird hinter einer bestimmten Zeichenkette der angegebene Text eingefügt. Beispiel: Vorher lautet die Zeile: "Der Edit ist ein Programm."

```
Eingabe: a/ein /flexibles / Nun steht dort: Der Edit ist flexibles
Programm.
```

b (before)

Hier wird vor einer bestimmten Zeichenkette der angegebene Text eingefügt. Beispiel: Vorher lautet die Zeile: "Edit kann noch viel mehr!."

```
Eingabe: b/Edit/Der / Nun steht dort: Der Edit kann noch viel
mehr!
```

d (delete)

Die aktuelle Zeile wird vollständig entfernt. Dabei fällt die entsprechende Zeilennummer einfach weg. Der Text wird also nicht neu durchnumeriert. Außer der aktuellen Zeile kann auch jede andere Zeile durch die Angabe ihrer Zeilennummer gelöscht werden. Ganze Textpassagen können entfernt werden, indem man ihre Start- und Endzeilennummer eingibt. Zum Beispiel:

```
"d 10 100"
```

Die Zeilen 10 bis 100 werden gelöscht.

i (insert)

Durch die Eingabe von I teilt man Edit mit, daß die folgenden Zeilen vor die aktuelle Zeile eingefügt werden sollen. Erst die Eingabe eines Z in einer gesonderten Zeile beendet diesen Einfügemodus. Anschließend wird der Pufferinhalt ab der ersten neuen Zeile neu durchnumeriert. Als Beispiel: Vorher lautet der Text:

```
20. "Durch den Einfügemodus wird"
21. "alles viel zu kompliziert."
```

Die Zeile 21 sei nun aktuell, und es werde die folgende Eingabe gemacht:

```
i
eine vernünftige Arbeit
mit EDIT erst möglich.
Man darf nur nicht denken, das sei
z
```

Das Ergebnis sieht dann so aus:

```
20. "Durch den Einfügemodus wird"
21. "eine vernünftige Arbeit"
22  "mit Edit erst möglich."
23. "Man darf nur nicht denken, das sei"
24. "alles viel zu kompliziert."
```

Arbeiten mit mehreren Dateien

Man kann jederzeit von Edit aus zusätzlich zu der beim Start angegebenen Datei noch weitere Quelldateien öffnen. Der Befehl hierzu lautet:

```
from .datei.
```

Nach dieser Eingabe werden neue Zeilen nur noch aus der Datei mit dem Namen datei geholt. Wie bei der ursprünglichen Quelldatei beginnt bei gerade geöffneten Dateien die Eingabe immer mit der ersten Zeile des Textes.

Durch *from* ohne Parameter gelangt man wieder zu der beim Start angegebenen Quelldatei. Das Programm läßt aber grundsätzlich alle Kanäle geöffnet, und merkt sich bei allen Dateien, wie viele Zeilen jeweils schon gelesen worden sind. Erst nachdem man eine Datei durch den Befehl cf .datei. (CloseFile) wieder geschlossen hat und sie erneut mit *from .datei.* öffnet, können schon einmal gelesene Zeilen ein weiteres Mal in den Puffer geholt werden.

Befehlsmakros

Ganz zu Beginn wurde schon erwähnt, daß Edit seine Befehle nicht nur einzeln von der Tastatur entgegennehmen kann. Man

kann ihn sogar mit einer Datei füttern, die ganz normale Edit-Befehle enthält. Den Namen dieser Datei gibt man entweder beim Start des Programms hinter dem Zusatz *With* an, oder man benutzt bei der Arbeit im Edit hierfür das Kommando C. Eine solche Makrodatei könnte z.B. folgendermaßen aussehen:

```
i
*****************************************************************
*  Programm            :                                        *
*  Autor               :                                        *
*  Datum               :                                        *
*  Sprache             : "C"                                    *
*  Assembler/Compiler  : Aztec C68/am-c V3.4                    *
*****************************************************************
z
```

Um diesen Vorspann nun vor die aktuelle Zeile im Edit einzufügen, genügt die Eingabe von C gefolgt von dem File-Namen der Datei, der von zwei Punkten begrenzt werden muß. Beispiel:

```
c .vorspann.
```

Das Einfügen wird nun nicht etwa durch das C-Kommando veranlaßt, vielmehr arbeitet Edit einfach nur die obige Datei ab. Hierbei taucht aber zuerst das Kommando I auf, wodurch der folgende Text bis zu dem Z (ausschließlich!) vor die aktuelle Zeile des Edit eingefügt wird. Sobald das Ende dieser Datei erreicht wird oder eine Zeile mit dem Befehl Q auftaucht, wird zur übergeordneten Befehlsebene zurückgekehrt. Dies muß nicht unbedingt wieder die Tastatur sein, da Befehlsdateien auch geschachtelt werden dürfen. In solchen Fällen steht also innerhalb der Befehlsdatei an einer Stelle wieder das C-Kommando. Man kann nun für alle erdenklichen Fälle seine eigenen Makrodateien konstruieren. Hat man den Bogen erst einmal raus, kann man mit Edit in einigen Fällen sogar schneller arbeiten als mit dem ED.

Beenden der Arbeit

Durch die Eingabe des Befehls W wird die Arbeit mit dem Edit im Normalfall beendet. Der Inhalt des Puffers und der eventuell vorhandene Rest der Quelldatei wird noch in die Zieldatei kopiert. Anschließend wird noch der ursprüngliche Text im Unter-

verzeichnis T unter dem Namen Edit-Backup eingetragen. Hat man bei Start des Edit nicht ausdrücklich den Namen einer Zieldatei angegeben, bekommt nun noch die Arbeitsdatei im Unterverzeichnis T, die ja als Zieldatei fungiert hat, den Namen der Quelldatei zugewiesen. Anschließend werden alle noch offenen Kanäle geschlossen, und das Programm wird verlassen.

Durch den Befehl Stop kann Edit auch verlassen werden, ohne daß diese Kopiervorgänge noch vorgenommen werden. In diesem Fall bleibt die Zieldatei natürlich unvollständig. Ist beim Start des Edit keine Zieldatei angegeben worden, wird nach Stop auch keine Umbenennung der Arbeitsdatei vorgenommen. Die Quelldatei ist also unverändert und unter dem gleichen Namen vorhanden.

Mit diesen Informationen zu dem Programm Edit kann schon sehr effektiv gearbeitet werden. Im Anhang finden Sie jedoch zusätzlich zu den behandelten Kommandos noch eine kurze Beschreibung der Befehle, die Edit außerdem noch zu bieten hat.

3. Die Standard-Devices des Amiga

Ein Device ist nichts anderes als ein Gerät, das mit dem Rechner Daten austauschen kann. Ein typisches Device ist also das Diskettenlaufwerk. Dieser Datenaustausch muß aber nicht immer in beide Richtungen erfolgen. Ein Drucker zum Beispiel nimmt nur Daten entgegen, während die Maus nur Daten an den Rechner schicken kann.

Eine Liste der Geräte, die vom CLI aus angesprochen werden können, findet man nach Eingabe des Befehls Assign. Die Standard-Devices eines jeden Amiga sind:

```
DF0 RAM PAR SER PRT CON RAW
```

Damit das CLI sie von Verzeichnissen oder Dateien unterscheiden kann, muß immer ein Doppelpunkt an ihren Namen gehängt werden.

Die folgenden Abschnitte zeigen, was es mit den einzelnen Bezeichnungen auf sich hat und wie man mit den Devices umzugehen hat.

3.1 Die Floppy-Devices

Alle Device-Namen, die mit der Kennung DF beginnen, beziehen sich auf die Floppylaufwerke des Amiga. Insgesamt können maximal vier solcher Stationen angeschlossen werden (DF0 - DF3). Entsprechend oft ist diese Kennung dann auch in der Assign-Liste vertreten. In der Grundkonfiguration mit nur einem Laufwerk beziehen sich alle Befehle des CLI - wenn nicht anders angegeben - automatisch auf das Laufwerk mit der Nummer Null. Die Probleme, die die Handhabung der verschiedenen Verzeichnisse eines Laufwerkes mit sich bringt, werden ausführlich im Kapitel 2 bei den Befehlen List und CD behandelt.

3.2 Das RAM-Disk-Device

Das Device mit dem Namen RAM simuliert ein Laufwerk im Arbeitsspeicher des Amiga. Das Wörtchen RAM ist eine Abkürzung für Random Access Memory, eine Speicherart, die einen wahlfreien Zugriff (Lesen & Schreiben) erlaubt. Bis auf wenige Ausnahmen kann man die RAM-Disk genauso ansprechen wie jedes andere Laufwerk auch. Der große Vorteil gegenüber ihren mechanischen Kollegen liegt in der hohen Geschwindigkeit, mit der Daten ausgetauscht werden können. Dieser Vorteil muß aber - wie sollte es anders sein - mit einigen Nachteilen erkauft werden.

Natürlich bleibt der Inhalt der RAM-Disk nur bis zum Ausschalten oder Neustart des Rechners erhalten. Da die Meldung Guru-Meditation - der totale Absturz des Rechners also - auch einen Neustart erforderlich macht, sollte man wichtige Daten von Zeit zu Zeit zusätzlich auf einer echten Floppy abspeichern, bevor man mit ihnen in der RAM-Disk weiterarbeitet.

Ein weiterer, entscheidender Nachteil ist der Speicherbedarf einer solchen RAM-Disk. Je mehr Daten in ihr abgelegt werden, desto weniger Speicher hat der Anwender für seine Arbeit zur Verfügung. Anders als bei den echten Disketten wächst die Speicherkapazität der RAM-Disk dynamisch. Je mehr Systemspeicher Sie zur Verfügung haben, desto mehr Daten kann auch die RAM-Disk aufnehmen. Brauchbare Auslastungsinformationen kann man von der RAM-Disk nicht erwarten. Sie gibt immer an, zu 100% belegt zu sein. Diese Angabe stimmt ja auch, da ihr vom System nicht mehr Speicher zugebilligt wird, als sie gerade benötigt.

Um mit der RAM-Disk arbeiten zu können, muß sie zuerst einmal erzeugt werden. Hierzu genügt es schon, sich einfach ihr Inhaltsverzeichnis mit *Dir RAM:* anzusehen. Meistenteils befindet sich dieser Befehl in der Startup-Sequence, so daß die RAM-Disk sofort benutzt werden kann. Ist sie erst einmal installiert, wird man sie so leicht auch nicht wieder los, denn einen Befehl DeleteRamDisk, gibt es (noch) nicht. Wie man sich bei der Arbeit mit nur einem angeschlossenen Laufwerk durch

Benutzung der RAM-Disk das häufige Diskettenwechseln ersparen kann, wird ausführlich im Abschnitt 3.5 besprochen.

Die nun folgenden Befehle dürfen zwar auf normale Laufwerke angewandt werden, nicht aber auf die RAM-Disk. Genauere Informationen zu den einzelnen Befehlen finden Sie im zweiten Kapitel.

AddBuffers

Dieser Befehl erzeugt unabhängig von der noch vorhandenen Speicherkapazität die Fehlermeldung:

```
Warning Insufficient memory for buffers
```

Es wäre natürlich reine Speicherverschwendung, wenn man der RAM-Disk zusätzlich noch einen Pufferspeicher innerhalb des RAM zuweisen könnte. Eine noch kürzere Zugriffszeit als bei der RAM-Disk ist wohl kaum möglich.

DiskCopy

Die Dateien oder Verzeichnisse einer RAM-Disk können nur einzeln mit dem Befehl Copy kopiert werden. Ein Kopieren der gesamten Disk mittels DiskCopy ist nicht möglich.

Format

Der Amiga merkt es zwar spät, aber er merkt es: Die RAM-Disk muß selbstverständlich vor der ersten Benutzung nicht formatiert werden. Versucht man es doch, erscheint die Fehlermeldung:

```
system/format failed -
not a disk
```

Relabel

Einen anderen Namen kann man der RAM-Disk leider nicht geben. Den Versuch beantwortet der Befehl mit:

```
Attempt to rename disk failed
```

Install

Dieser Befehl macht normale Disketten bootfähig. Mit ihnen kann dann der Amiga gestartet werden. Daß die RAM-Disk für einen Neustart denkbar ungeeignet ist, dürfte sich von selbst verstehen. Der Befehl Install funktioniert daher hier auch nicht.

3.3 Das parallele Device

Über den Device-Namen PAR kann eine parallele Schnittstelle vom Typ Centronics angesprochen werden. Damit ist genau der Anschluß gemeint, der beim Amiga unter dem Namen Parallel Port aus dem Gehäuse geführt wird. Im Gegensatz zu den schon behandelten Devices muß hier das eigentliche Gerät erst noch angeschlossen werden. Die Schnittstelle ist deswegen parallel, weil immer gleichzeitig alle acht Bits eines Bytes übertragen werden. Es wäre natürlich auch möglich, die Informationen eines Bytes nacheinander, also Bit für Bit zu übertragen. Ein solches Device wird im nächsten Abschnitt behandelt.

Die Schnittstelle ist keineswegs auf irgendein bestimmtes Gerät fixiert. Beispielsweise könnte hier ein Analog-/Digital-Wandler angeschlossen werden, mit dem kontinuierliche Signale (Sprache, Musik, Meßwerte) auf ein für den Rechner verständliches Format gebracht werden können. In diesem Fall würden also Daten von außen über die Schnittstelle zum Rechner geschickt. Aber auch die andere Datenflußrichtung ist möglich. Eine typische Anwendung hierfür ist der Anschluß eines Druckers an den Amiga. Die eigentlichen Informationen laufen dabei vom Rechner über die Schnittstelle zu dem Drucker. Warum man bei einem Drucker mit paralleler Schnittstelle in den meisten Fällen jedoch das Prt-Device benutzen muß, wird bei der Behandlung dieses Devices begründet.

Die Geschwindigkeit, mit der die Datenübertragung stattfindet, hängt alleine davon ab, wie schnell die Daten von den Geräten verarbeitet werden können. Zusätzlich zu den acht Leitungen, die für die Übertragung der Nutzdaten zuständig sind, gibt es bei einer Centronics-Schnittstelle noch einige Steuerleitungen, die für den sogenannten Handshake zuständig sind. Über diese

Leitung kann z.B. ein Daten empfangendes Gerät (Drucker) dem sendenden Gerät (Amiga) mitteilen, daß es bereit ist, neue Daten zu empfangen. Hierdurch wird immer eine optimale Übertragungsgeschwindigkeit gewährleistet.

3.4 Das serielle Device

Hinter dem Namen SER versteckt sich eine serielle Schnittstelle vom Typ RS 232C. Der zugehörige Anschluß am Amiga heißt Serial Port und kann für die verschiedensten Anwendungen (Modem, MIDI etc.) verwendet werden. Der Name verrät die besonderen Eigenschaften dieses Schnittstellentyps: Die einzelnen Bits eines Bytes werden nacheinander und nicht wie bei dem PAR-Device gleichzeitig übertragen. Daher wird für jede Datenrichtung auch nur eine Signalleitung benötigt. Bei Bedarf können auch jeweils nur die untersten sieben Bits eines Bytes übertragen werden. Das freigewordene Bit kann dann zur Übertragungskontrolle mitgesendet werden (Paritäts-Bit). Je nachdem, wie die Parität vor Benutzung der Schnittstelle gewählt wird (gerade oder ungerade), wird dieses achte Bit vom Sender automatisch so gesetzt, daß entweder alle gesetzten Bits zusammen immer eine gerade oder aber immer eine ungerade Zahl ergeben (Even Parity bzw. Odd Parity). Der Empfänger muß natürlich auf die gleiche Parität eingestellt sein und kann so in beschränktem Maße Übertragungsfehler feststellen.

Zur Synchronisation wird jede Übertragungseinheit noch um ein Start-Bit und ein bis zwei Stop-Bits erweitert. Die Geschwindigkeit, mit der die einzelnen Bits übertragen werden, muß beim sendenden und empfangenden Gerät genau übereinstimmen. Sie wird traditionell in Baud (nach dem franz. Erfinder Baudot) gemessen. Ein Baud entspricht der Übertragungsgeschwindigkeit von einem Bit pro Sekunde.

Das Handshake-Problem taucht natürlich auch bei der RS232-Schnittstelle auf. Hier gibt es insgesamt drei mögliche Verfahren, wie eine korrekte Datenübernahme gesichert werden kann:

Senden bestimmter Kommando-Bytes

Dieses Verfahren wird meist xON/xOFF-Protokoll genannt. Voraussetzung für die Anwendung dieses Verfahrens ist eine bidirektionale Beschaltung der Schnittstelle. Sobald ein Gerät aus irgendeinem Grund keine weiteren Daten empfangen kann, sendet es auf der Rückleitung ein spezielles Steuerzeichen (xOFF). Das sendende Gerät unterbricht jetzt solange die Datenübertragung, bis es auf dieser Leitung das xON-Zeichen erhält. Der Vorteil dieser Methode liegt darin, daß man mit nur drei Leitungen auskommt (2 Signalleitungen, 1 Masse), weshalb häufig auch von einem 3-Draht-Handshake gesprochen wird. Das Betriebssystem sorgt bei entsprechender Programmierung automatisch für die korrekte Auswertung der Steuerzeichen.

Benutzung von Steuerleitungen

Diese Methode ist vergleichbar mit dem Handshake der parallelen Schnittstelle. Es werden noch zusätzliche Leitungen benötigt (RTS/CTS bzw. DSR/DTR), mit denen die Geräte Informationen über ihren Zustand austauschen können. Ein eigenständiger Datenkanal nur für die Rückmeldungen kann hierbei entfallen. Der Vorteil dieser Methode liegt in der insgesamt etwas schnelleren Übertragungsgeschwindigkeit, da die Steuerkommandos nicht über den relativ langsamen Datenkanal laufen.

Sicherheitsreserven

Wenn man sich 100% sicher sein kann, daß ein datenempfangendes Gerät die ankommenden Bytes schneller verarbeiten kann, als sie der Sender auf die Leitung legt, kann auf den Handshake völlig verzichtet werden. Diese Methode bietet sich an, wenn man schnell einmal mit möglichst wenig Aufwand (2 Leitungen) Daten von einem Rechner auf einen anderen übertragen will. Hierzu muß durch eine geeignete Warteschleife auf Seiten des Senders genügend Luft für den Empfänger erzeugt werden. Für eine dauerhafte Lösung ist dieses Verfahren aber zu uneffektiv, da zuviel Zeit verschwendet wird.

Alle besprochenen Parameter müssen vor der ersten Benutzung der seriellen Schnittstelle mit dem Programm Preferences einge-

stellt werden. Zusätzlich findet man hier noch ein Feld mit der Überschrift Buffer Size, mit dem man die Größe des Zwischenspeichers für die Empfangsdaten festlegen kann. Dieser Puffer stellt den Datenempfang sicher, falls das Leseprogramm kurzzeitig nicht aufnahmebereit ist. Werden die Daten jedoch zu langsam gelesen und es setzt kein Handshake ein, kann der Puffer überlaufen. Daten, die jetzt noch eintreffen, sind verloren.

Bei unseren Versuchen mit der seriellen Schnittstelle SER waren wir das erste Mal richtig enttäuscht vom Amiga. Von der Vielzahl an Parametern, die das Amiga-Betriebssystem für eine serielle Schnittstelle zur Verfügung stellt, kann nur ein Bruchteil über das Programm Preferences eingestellt werden. Gibt man den Befehl *Copy SER: to con:10/10/300/100/Ausgabe* im CLI ein, wird solange gewartet, bis der RS232-Eingabepuffer (mind. 512 Bytes) vollgeschrieben ist. Erst dann erscheinen die ersten empfangenen Daten in dem Con-Fenster. Die dann noch im Puffer verbliebenen Daten werden erst ausgegeben, wenn erneut ein längeres Datenpaket übertragen wird. Eine Endekennung kann also nicht übertragen werden. Auf diesem Gebiet ist von den Programmierern des Programms Preferences keine überzeugende Leistung erbracht worden.

Die interessanten Erfahrungen, die wir bei der Verbindung zweier Amiga-Rechner via RS232 gemacht haben, schildern wir Ihnen ausführlich im Kapitel 3.8.

3.5 Das Drucker-Device

Speziell für Ausgaben an den Drucker gibt es noch das Printer-Device. Programmierer, die den Drucker ansteuern möchten, müssen sich keine Gedanken darüber machen, ob der Drucker nun an der seriellen oder an der parallelen Schnittstelle angeschlossen ist. Wird das Prt-Device angesprochen, zählen in jedem Fall die Angaben, die in dem Programm Preferences gemacht wurden. Da man hier auch seinen speziellen Druckertyp angeben muß, können alle Programme für bestimmte Druckerfunktionen einheitliche Steuerzeichen verwenden. Denn nicht alle Drucker benutzen zur Aktivierung bestimmter Funktionen den gleichen

Code. Trotz der vielen Druckertreiber, die auf der Workbench-Diskette vorhanden sind, gibt es immer wieder Schwierigkeiten mit einigen Druckern.

Wir sind daher der Meinung, daß die Methode mit den fertigen Druckertreibern ganz einfach verkehrt ist. Viel besser wäre ein vernünftiges Preference-Programm, das es ermöglicht, für jeden Drucker seinen individuellen Treiber selber zusammenzustellen. Die paar Möglichkeiten, die Preferences im Augenblick zur Verfügung stellt, reichen hierfür natürlich längst nicht aus.

3.6 Das Con-Device

Da die Tastatur und der Bildschirm des Amiga im Grunde genommen ganz normale Ein- und Ausgabegeräte sind, kann man sie wie jedes andere Device auch ansprechen. Sowohl Eingaben als auch Ausgaben finden dabei in einem eigenen Fenster statt. Das Console-Device muß folgendermaßen angesprochen werden:

```
con:x/y/breite/höhe/name
```

Die Angaben hinter Con: haben die folgenden Bedeutungen:

x/y

Koordinaten der linken oberen Ecke des Fensters.

breite

Breite des Fensters in Pixeln.

höhe

Höhe des Fensters in Pixeln.

name

Beliebiger Name, der oben im Fenster erscheint. Beispiel 1:

```
dir >con:10/10/300/100/Testfenster
```

Hierdurch wird die Ausgabe des Directorys in ein eigenes Fenster mit den angegebenen Ausmaßen umgeleitet. Sobald die Ausgabe beendet ist, verschwindet das Fenster wieder. Beispiel 2:

```
copy con:10/10/300/100/Eingabe con:10/150/300/100/Ausgabe
```

Es erscheinen gleich zwei Con-Fenster auf dem Bildschirm. Die Eingaben, die im oberen Fenster gemacht werden, zeigt nach einem Return auch das untere Fenster. Die Eingabe von Ctrl und Backslash (\) läßt beide Fenster wieder verschwinden.

3.7 Das Raw-Device

Dieses Device ist eng verwandt mit dem Console-Device. Es wird genauso angesprochen und verhält sich auch auf den ersten Blick gleich. Den ersten Unterschied wird man bei Eingaben in dieses Fenster feststellen. Es erscheinen hier keine Zeichen. Zur Demonstration der Funktion ist das folgende Beispiel sehr gut geeignet:

```
copy raw:10/10/300/100/Eingabe con:10/150/300/100/Ausgabe
```

Geben Sie nun einmal im oberen Raw-Fenster Zeichen ein. Anders als bei einem Con-Fenster wird jedes sofort übertragen. Es wird also nicht erst auf die Eingabe der Return-Taste gewartet. Wird diese jedoch gedrückt, erscheint der Cursor wieder am Anfang der Zeile.

Sehr erfreulich ist die Tatsache, daß sämtliche Steuerzeichen wie z.B. Cursor, Delete und Backspace übertragen werden können. Es ist die Aufgabe des Empfängers (hier also des Con-Devices), diese Sonderzeichen korrekt auszuwerten. In unserem Beispiel funktionieren nur die Cursor-Tasten so, wie man es gewohnt ist. Die Eingabe von Ctrl und C beendet den ganzen Spuk wieder.

Sollte die Ausgabe nicht funktionieren, liegt das mit aller Wahrscheinlichkeit daran, daß Sie zuerst im unteren Fenster Eingaben gemacht haben. Wie im CLI-Fenster wird dadurch die Ausgabe anderer Daten unterdrückt. In diesem Fall sollten Sie im unteren Fenster einmal die Return-Taste drücken.

4. Workbench und Kickstart 1.3

Die Entwickler des Amiga haben sich in der letzten Zeit keinesfalls auf ihren Lorbeeren ausgeruht. Die seit einiger Zeit fertiggestellte Version 1.3 der Amiga Firmware (Kickstart bzw. Workbench) weist Verbesserungen und Erweiterungen auf, die man schon in kürzester Zeit nicht mehr missen möchte. Um nur einige Punkte zu nennen:

- Neue bzw. erheblich verbesserte CLI-Befehle.
- Komfortable Shell anstelle des CLI.
- CLI-Befehle können resident geladen werden.
- Schnelleres File-System für Laufwerke mit nicht auswechselbarem Speichermedium (Festplatten, RAM-Disk).
- Schnellere Mathe-Bibliothek.
- Neue Geräte-Handler (Aux, Speak, Pipe, NewCon).
- Resetfeste RAM-Disk, von der bei Verwendung des Kickstart 1.3 auch gebootet werden kann.
- Boot-Möglichkeit von speziellen Geräten (nur Kickstart 1.3).

Wie man der Liste entnehmen kann, funktioniert ein Großteil der Neuerungen auch bei Verwendung von Kickstart 1.2. Dies wird besonders die Besitzer eines Amiga 500 bzw. 2000 erfreuen, da bei diesen Rechnern das Kickstart fest in ein ROM (engl. Read Only Memory = Nur Lesespeicher) eingebrannt ist und daher eine neue Version des Betriebssystems nicht, wie es beim Amiga 1000 der Fall ist, von einer Diskette nachgeladen werden kann.

Im wesentlichen entfällt bei der Kombination Workbench 1.3/ Kickstart 1.2 die Möglichkeit des Bootens von speziell dafür ausgelegten Geräten. Zu diesen Geräten zählt auch die neue RAM-Disk RAD, von der aus der Amiga bei Verwendung von Kickstart 1.3 in Sekundenschnelle neu gebootet werden kann. Es besteht also für Benutzer von Kickstart 1.2 auch weiterhin nur die Möglichkeit, von einer Diskette zu booten, die sich im Laufwerk Null befindet.

Größter Wert wurde bei allen Änderungen auf die Kompatibiliät zur alten Workbench- bzw. Kickstart-Version gelegt. Daher dürfte es nur in den wenigsten Fällen zu Unverträglichkeiten mit vorhandener Software kommen. Sollten sich dennoch Schwierigkeiten ergeben (wir haben zum Beispiel Probleme mit dem Debugger db des Aztek-Compilers feststellen müssen), so sollte man auf die alte Workbench zurückgreifen.

Kommen wir nun zur Beschreibung der neuen Funktionen.

4.1 Neue CLI-Befehle der Workbench 1.3

Im C-Verzeichnis der Workbench-Diskette findet man einige sehr nützliche Befehle, die neu hinzugekommen sind. Mit diesen Befehlen wollen wir uns jetzt näher beschäftigen.

4.1.1 Avail

Syntax: `avail`

Bisher war man auf die Angaben in der Workbench-Titelzeile angewiesen, wenn man sich über den Belegungsgrad des Arbeitsspeichers informieren wollte. Dank des neuen Avail-Befehls ist dies nun nicht mehr notwendig. Avail leistet sogar noch mehr, wie es die folgende Beispielausgabe demonstriert:

```
Type  Available    In-Use    Maximum   Largest
chip      77472    445760     523232     42712
fast     226200    290688     516888    219008
total    303672    736448    1040120    219008
```

In jeder Spalte werden Angaben für das CHIP-Memory, das FAST-Memory (nur bei einer Speicher-Erweiterung vorhanden) und für den gesamten Speicherbereich (CHIP-Mem und FAST-Mem) gemacht. Unter Available findet man den derzeit noch verfügbaren Speicherplatz, während In-Use Angaben über die Größe des belegten Speichers macht. Beide Werte zusammengenommen ergeben den Wert, den man unter Maximum findet.

Von Interesse ist häufig auch die Angabe über den Umfang des größten, zusammenhängenden Speicherbereichs, denn ein freier Speicher von 1 MByte ist völlig nutzlos, falls er aus lauter nur 1 KByte großen Stücken besteht. Ein Programm kann sich zwar beim Laden auf mehrere kleine Speicherbereiche aufteilen, jedoch müssen diese Speicherbereiche eine bestimmte Mindestgröße aufweisen. Sollte sich ein Programm nicht mehr laden lassen, obwohl es laut List-Befehl noch in den Speicher passen müßte, liegt das mit großer Wahrscheinlichkeit an einem zu zerklüfteten Arbeitsspeicher.

4.1.2 FF

Hinter der Bezeichnung FF verbirgt sich ein Programm mit dem Namen FastFonts. Es ist von der Firma Microsmiths entwickelt worden und schon seit einiger Zeit auf dem Markt erhältlich. FastFonts beschleunigt die Ausgabe von Texten auf dem Amiga. Man darf jedoch nicht allzuviel von dem Programm erwarten, denn die Ausgaben werden maximal nur um etwa 20% schneller.

Durch die Eingabe von *ff -0* wird FastFonts aktiviert. Es erscheint die Meldung:

```
FastFonts V1.1 Copyright - 1987 by C.Heath of Microsmiths, Inc
Turning on FastText
```

Man findet den Befehl normalerweise innerhalb der Startup-Sequence einer Boot-Diskette wieder. Hier wird jedoch die Einschaltmeldung von FastFonts unterdrückt, indem sie an das NIL-Device umgelenkt wird: *ff >nil: -0*.

Sollte aus irgendeinem Grunde wieder der normale Ausgabemodus benötigt werden, genügt die Eingabe des Befehls ff -n. Auch hierbei erscheint die Startmeldung von FastFonts. Die Meldung "Turning on FastText" unterbleibt dabei jedoch.

4.1.3 Lock

Syntax: `DRIVE/A,ON/S,OFF/S,PASSKEY`

Dieser neue Befehl funktioniert nur in Verbindung mit einer Harddisk, deren Partitions unter dem neuen FastFilingSystem (FFS) der Workbench 1.3 laufen. Lock bietet die Möglichkeit, beliebige Partitions mit einem Schreibschutz zu versehen. Eine solche Partition verhält sich anschließend ähnlich wie eine Diskette, deren Schreibschutzclip auf *Write Protect* steht.

Zusätzlich bietet der Befehl Lock aber noch die Möglichkeit, den Schreibschutz-Zustand mit einem beliebigen Paßwort zu sichern. Damit ist ein Aufheben des Schreibschutzes nur möglich, falls man über das Paßwort informiert ist:

```
lock dh1: on beethoven
```

Jeder Schreibversuch auf die Partition dj1: wird jetzt vom DOS mit der Meldung "Volume xxx is write protected" abgewiesen (xxx steht hierbei für den Namen der Partition dh1:).

Erst nach Eingabe des Befehls *lock dh1: off beethoven* werden Schreibzugriffe wieder ausgeführt.

4.1.4 NewShell

Syntax: `Window,From`

Mit Hilfe von NewShell kann ein weiteres Fenster für die Eingabe von DOS-Befehlen geöffnet werden. Früher geschah dies mit Hilfe des NewCLI-Befehles. Eine Shell weist jedoch gegenüber dem CLI die folgenden Vorteile auf:

Die Eingabezeile kann mit Hilfe der Cursor-Tasten beliebig ediert werden

Die Shell benutzt für Ein- und Ausgaben das NewCon-Device. Diese neue Fensterschnittstelle ist für einen großen Teil der

neuen Shell-Eigenschaften verantwortlich. NewCon erlaubt im Gegensatzt zu dem Con-Device die Benutzung der Cursor-Tasten, um einen eingegebenen Text verändern zu können. Der Cursor kann mit den Tasten Pfeil links und Pfeil rechts beliebig auf der Eingabezeile plaziert werden. Selbstverständlich funktionieren die Tasten Delete (Del) und Backspace (Pfeil nach links), wie man es von ihnen gewohnt ist. Zusätzlicher Text wird vor der aktuellen Cursor-Position eingefügt. Erst wenn die Return-Taste betätigt wird, übernimmt die Shell die eingegebene Zeile.

Zusätzlich werden die folgenden Tastenkombinationen ausgewertet:

Shift Cursor links (oder Ctrl-A)

Setzt den Cursor auf den Anfang der Zeile.

Shift Cursor rechts (oder Ctrl-Z)

Setzt den Cursor auf das Ende der Zeile.

Ctrl-K

Löscht Text von der Cursor-Position bis zum Ende der Zeile.

Ctrl-U

Löscht den Text links neben dem Cursor.

Ctrl-W

Setzt den Cursor auf die nächste Tabulator-Position.

Ctrl-X

Löscht die gesamte Zeile.

Schon einmal eingegebene Befehle können mit den Cursor-Tasten zurückgeholt werden

Auch diese neue Eigenschaft verdanken wir dem NewCon-Device. Alle eingegebenen Befehle werden in einem 2 KByte großen Ringpuffer zwischengespeichert. Mit Hilfe der Taste Pfeil hoch kann man auf die zuletzt eingegebenen Befehle zurückgreifen. Sehr nützlich ist diese Funktion, wenn ein Befehl aufgrund eines Schreibfehlers nicht ausgeführt worden ist. Mit einem Tastendruck hat man die fehlerhafte Befehlszeile wieder vor sich und kann sie nun mit Hilfe der Cursor-Tasten schnell korrigieren.

Sucht man einen speziellen Befehl, dessen Eingabe schon einige Zeit zurückliegt, kann einem das NewCon-Device bei der Suche helfen: Es genügt, wenn man von dem Befehl nur die ersten Buchstaben eingibt und anschließend die Taste Pfeil hoch in Verbindung mit Shift betätigt. Es wird der Befehl gesucht, auf den die Beschreibung zutrifft.

An das Ende des Puffers gelangt man durch die Tastenkombination Shift + Pfeil nach unten.

Steuerzeichen (Ctrl-Codes) werden neutral behandelt

Gibt man in der Shell ein Steuerzeichen ein (z.B. *Ctrl L*), wird dies nicht ausgegeben. Statt dessen erscheint an der aktuellen Cursor-Position nur ein Leerzeichen. Das Steuerzeichen ist aber dennoch vorhanden und kann ganz normal verwendet werden.

Ein Startup-Batchfile wird ausgeführt

Bei jedem Aufruf des NewShell-Befehls wird automatisch eine Batch-Datei mit dem Namen Shell-Startup aufgerufen. Diese Datei befindet sich im Verzeichnis s der Workbench-Diskette und kann Befehle aufnehmen, die eine bestimmte Grundkonfiguration der Shell vornehmen. Hier kann zum Beipiel festgelegt werden, wie das Shell-Prompt aussehen soll. Wie man mit Batch-Dateien umzugehen hat, wird ausführlich in einem eigenen Kapitel dieses Buches behandelt.

Die nun folgenden Shell-Funktionen sind nur verfügbar, falls vor dem Aufruf der Shell das sogenannte Shell-Segment in das Betriebssystem eingebunden worden ist. Die Befehlsfolge, die hierzu notwendig ist, lautet *resident CLI l:Shell-Seg SYSTEM.* Diesen Befehl findet man normalerweise in der Startup-Sequence der Workbench-Diskette wieder, so daß eine manuelle Eingabe nicht notwendig ist.

Residente Befehle können aufgerufen werden

Nur von einer Shell aus können Programme aufgerufen werden, die mit Hilfe des Befehls Resident in den Arbeitsspeicher des Amiga geladen worden sind. Diese Befehle stehen dem Anwender sofort zur Verfügung und müssen nicht erst von einem Laufwerk geladen werden. Nähere Informationen zu diesem Thema finden Sie bei der Behandlung des Resident-Befehls.

Programmnamen können beliebig abgekürzt werden

Man sollte sich davor hüten, den AmigaDOS-Befehlen, die sich im C-Verzeichnis der Workbench-Diskette befinden, einfach einen anderen Namen zu geben. Benennt man zum Beispiel den Fault-Befehl mittels Rename um in FEHLER, so werden Fehlermeldungen des DOS nicht mehr im Klartext ausgegeben, sondern es erscheint nur noch die jeweilige Fehlernummer. Viele Programme verwenden die Befehle des AmigaDOS, die aus diesem Grunde nicht umbenannt werden dürfen.

Die AmigaShell ist jedoch mit einer Funktion ausgestattet, die es ermöglicht, einen Befehl zusätzlich unter einem (oder mehreren) beliebigen Namen aufrufen zu können. Dieser beliebige Name muß vorher mit Hilfe des Alias-Befehls festgelegt werden. Ein Beispiel:

```
alias ex execute
```

Nach Eingabe dieser Zeile kann der Execute-Befehl auch unter dem Namen *ex* aufgerufen werden. Der Alias-Befehl ordnet der ersten Zeichenkette (hier: ex) den gesamten Text der restlichen

Zeile zu. In diesem Fall besteht der Rest der Zeile nur aus dem Wort *execute*. Es sind aber auch solche Abkürzungen erlaubt:

```
alias st-up ed s:startup-sequence
```

Es genügt nun, den Befehl *st-up* einzugeben, um die Startup-Sequence in den ED zu laden.

Möchte man die Alias-Befehle nicht nach jedem Öffnen einer Shell neu eingeben, so kann man sie auch in die Batch-Datei Shell-Startup aufnehmen. Wie schon gesagt, wird diese Batch-Datei bei jedem NewShell automatisch ausgeführt.

Das Kommando *alias* ohne zusätzlichen Angaben listet sämtliche derzeit vorhandenen Namenszuordnungen auf.

4.1.5 Remrad

Syntax: `remrad`

Mit Hilfe des Kommandos Remrad (Remove Recoverable Ram Disk) kann der Inhalt der resetfesten Ram-Disk RAD: gelöscht werden. Die RAM-Disk belegt anschließend nur noch einen relativ kleinen Speicherplatz. Beim nächten Neustart des Rechners wird auch dieser Speicher wieder dem System zugeführt.

4.1.6 Resident

Syntax: `resident NAME,FILE,DELETE/S,ADD/S,REPLACE/S,PURE/S,SYSTEM/S`

Auf diesen Befehl haben wohl alle Besitzer eines Amiga schon lange gewartet. Bisher mußten CLI-Befehle oder Programme vor ihrer Ausführung grundsätzlich erst von einem Laufwerk (Diskette, Harddisk oder RAM-Disk) geladen werden. Dadurch wurde besonders die Arbeit mit nur einem Floppy-Laufwerk auf dem Amiga nahezu unerträglich. Andauernd wurde man aufgefordert, die Workbench-Diskette in das Laufwerk einzulegen, obwohl sich der eingegebene Befehl oft auf eine ganz andere Diskette beziehen sollte (Beispiel: *Dir-Befehl*). Resident er-

möglicht es nun dem Anwender, seine am häufigsten benötigten Befehle fest in den Arbeitsspeicher des Amiga zu laden. Dadurch ist der Befehl jederzeit sofort ausführbar und muß nicht erst umständlich geladen zu werden.

Als es den Befehl Resident noch nicht gab, kopierte man die notwendigen CLI-Befehle meist in die RAM-Disk und teilte dem DOS durch den Path-Befehl mit, daß zuerst hier nach den Befehlen gesucht werden solle, bevor die Workbench-Diskette verlangt wird (s. Path-Befehl). Diese Methode funktioniert zwar ausgezeichnet, hat jedoch einen großen Nachteil: Ruft man einen Befehl auf, der sich in der RAM-Disk befindet, so wird er, wie bei jedem anderen Laufwerk auch, erst in den Speicher geladen. Bei den auch heutzutage noch relativ hohen Kosten für den RAM-Speicher ist dieses Verfahren viel zu verschwenderisch, da ja nun die Daten des Programms an zwei Stellen des Arbeitsspeichers vorhanden sind. Bei jedem neuen Aufruf des Programms entstehen weitere Kopien, die wertvolles RAM in Anspruch nehmen.

Befehle hingegen, die durch Resident geladen wurden, befinden sich nur ein einziges Mal im Arbeitsspeicher. Ruft man solche Befehle vom CLI aus auf, stehen sie unabhängig von ihrem Umfang ohne merkliche Verzögerung sofort zur Verfügung. Selbst wenn der Befehl von einem zweiten CLI aus ein weiteres Mal gestartet wird, befindet sich der Programmcode nur an einer Stelle im RAM.

Man kann sich leicht vorstellen, daß an einen CLI-Befehl einige Bedingungen gestellt werden müssen, damit dieses Verfahren überhaupt funktionieren kann:

1. Die Befehle müssen "re-executable" sein. Dies bedeutet nichts anderes, als daß man den Befehl von mehreren CLI-Fenstern aus gleichzeitig benutzen kann. Beipiel: Im CLI-Fenster 1 läßt man sich mittels *dir df0:* das Verzeichnis der Diskette in Laufwerk 0 ausgeben, während im CLI-Fenster 2 der Dir-Befehl auf die Diskette in Laufwerk 1 angewandt wird (*dir df1:*). Bis auf wenige Ausnahmen sind alle Programme (also nicht nur die CLI-Befehle) auf dem Amiga "re-executable".

2. Die Befehle müssen "re-entrant" programmiert sein. Wie oben schon beschrieben wurde, befindet sich der Programmcode eines residenten Befehls nur an einer Stelle des Arbeitsspeichers, auch wenn der Befehl mehrere Male gleichzeitig abläuft. Die wichtigste Eigenschaft, die einen re-entrant programmierten Befehl auszeichnet, ist daher die ausschließliche Verwendung lokaler Variablen, die bei jedem Programmaufruf neu angelegt werden müssen.

Zum Verständis dieses Problems stelle man sich ein Zeichenprogramm vor, das in einer Speicherstelle den Farbcode für die aktuelle Zeichenfarbe enthalte. Diese Speicherstelle sei direkt nach dem Laden des Programms mit dem Code für die Farbe Weiß vorbelegt. Ändert man nun beim Zeichnen diese Farbe z.B. auf Rot und startet anschließend das Zeichenprogramm ein weiteres Mal, so besitzt dieses zweite Programm nun die Farbe Rot anstelle von Weiß als Grundeinstellung, da es die gleichen Speicherstellen benutzt. Natürlich ist dies noch ein völlig harmloses Beispiel. Der Amiga macht hier jedoch keine Unterschiede und würde auf den zweiten Aufruf des residenten Zeichenprogramms mit einer Guru Meditation antworten.

Damit solche Rechnerabstürze verhindert werden können, hat man das neue Status-Flag P (für Pure) eingeführt. Dieses Flag kann wie die schon bekannten Flags RWED (s. List-Befehl) mit Hilfe des Protect-Befehls für jede Datei gesetzt bzw. gelöscht werden. Ein gesetztes Pure-Flag signalisiert dem DOS, daß das zugehörige Programm mit Hilfe des Resident-Befehls geladen werden darf. Sieht man sich die Liste der neuen CLI-Befehle einmal an, stellt man fest, daß bei allen Befehlen, bei denen es einen Sinn ergibt, das Pure-Flag gesetzt ist.

Kommen wir nun endlich zur Anwendung des Resident-Befehls. Gibt man den Befehl ohne Parameter ein, so erhält man eine Liste der Befehle, die derzeit resident vorhanden sind. Außerdem werden dabei die residenten System-Segmente angezeigt. Beispiel:

```
Name            UseCount
CD                 1
Dir                1
Execute            1
CLI             SYSTEM
FileHandler     SYSTEM
Restart         SYSTEM
CLI             SYSTEM
```

Unter UseCount findet man Informationen darüber, wie oft der jeweilige Befehl derzeit aktiv ist. Diese Angabe ist jedoch grundsätzlich um Eins zu hoch. Eine Eins bedeutet hier also, daß der Befehl zur Zeit nicht benutzt wird. Bei System-Segmenten steht unter UseCount der Kommentar SYSTEM.

In der Syntax-Liste des Resident-Befehls steht NAME bzw. FILE für die genaue Pfadangabe des Befehls oder Segments, der (das) resident gemacht werden soll. Beispiel:

```
resident c:dir
```

Nach einem kurzen Augenblick steht der Dir-Befehl in der Shell uneingeschränkt zur Verfügung.

Wendet man den Resident-Befehl auf eine Datei an, deren PURE-Flag nicht gesetzt ist, erhält man die Fehlermeldung:

```
Pure bit not set
Cannot load xxx
(xxx steht hierbei für den File-Namen)
```

Mit Hilfe des Zusatzes PURE kann man jedoch auch dann ein File mittels Resident laden, wenn das PURE-Flag nicht gesetzt ist. Sicherheitshalber wird auch in diesem Falle die Meldung "Pure bit not set" ausgegeben. Die PURE-Option sollte aber mit größter Vorsicht verwendet werden, da Programme, deren PURE-Flag nicht gesetzt ist, meist wirklich nicht re-entrant programmiert worden sind. Man darf sich daher über eine später auftretende Guru Meditation nicht wundern.

Die Option REMOVE dient dazu, einen Eintrag aus der Liste der residenten Files zu streichen. Beispiel:

```
resident execute remove
```

Der Befehl execute wird entfernt und der durch ihn belegte Speicherplatz wieder freigegeben.

Bei System-Segmenten wird der Wert von UseCount auf minus eins gesetzt. Da sich ein Eintrag jedoch nur entfernen läßt, falls UseCount auf eins steht, kann ein einmal eingetragenes Segment mit *remove* nicht wieder gelöscht werden.

Mit der Option ADD gelingt es, mehrere Befehle bzw. Segmente mit dem gleichen Namen resident zu machen. Es kann jedoch von der Shell aus immer nur der zuletzt eingegebene Befehl aufgerufen werden.

REPLACE dient dazu, einen beliebigen Befehl (oder ein beliebiges Segment) gegen einen schon in der Liste vorhandenen Befehl (ein vorhandenes Segment) zu ersetzen. Ist zum Beispiel der Befehl Execute derzeit resident und gibt man nun ein:

```
resident execute date replace
```

so kann man anschließend den Date-Befehl durch die Eingabe von Execute aufrufen. Diese Option ist aber eher dazu gedacht, System-Segmente unter Beibehaltung ihres Namens zu ersetzen.

4.1.7 SetAlert

Syntax: `setalert`

Diesen Befehl findet man in der Startup-Sequence der neuen Workbench-Diskette wieder. Er modifiziert das Kernal derart, daß nach einem Recoverable Alert nicht mehr notwendigerweise eine Guru Meditation folgt. SetAlert wird dazu mittels Run ein eigener Hintergrundprozeß eingeräumt. Davon kann man sich mit Hilfe des Status-Befehls überzeugen.

4.1.8 Setenv/Getenv

Syntax: `setenv NAME/A,String`
Syntax: `getenv NAME/A`

Diese beiden Befehle sind derzeit noch nicht voll funktionstüchtig. Sie sollen später die Verwendung sogenannter Environment-Variablen ermöglichen. Dazu fehlt jedoch noch der Environment-Handler, der sich in der Entwicklung befindet. Zur Zeit wird der Handler zwar durch die RAM-Disk simuliert, sinnvolle Anwendungen lassen sich aber damit noch nicht realisieren.

4.1.9 XIcon

Mit Hilfe dieses Befehls kann eine Batch-Datei jetzt auch von der Workbench aus durch einfaches Doppelklicken aufgerufen werden. Dazu muß bei geladener Workbench folgendermaßen vorgegangen werden:

- Erstellen eines PROJECT-Icons z.B. mit Hilfe des Icon-Editors der Extras-Diskette. Ein PROJECT-Icon erhält man, wenn man zum Beispiel das CLI-Icon auf der Workbench-Diskette lädt, beliebig modifiziert und unter dem Namen der Batch-Datei abspeichert. Die Option *Frame and Save* des Icon-Editors ermöglicht auch das Abspeichern kleinerer Icons, indem man den gewünschten Bildbereich einfach mit der Maus eingrenzt.

- Anschließend den Diskettenordner mit dem neuen Icon öffnen, das Icon einmal anklicken und den Info-Menüpukt der Workbench anwählen. Bei dem nun erscheinenden Info-Fenster muß in das Feld DEFAULT TOOL der Befehl *C:XIcon* eingetragen werden. Wird das Info-Fenster nun abgespeichert (Save), kann die Batch-Datei auch schon durch Doppeklicken aufgerufen werden. Bei Bedarf können jedoch in dem Feld TOOL TYPE des Info-Bildschirms noch Angaben über die Größe des Fensters für die Ausgaben des Batch-Files gemacht werden. Beispiel:

```
TOOL TYPE  WINDOW=CON:0/0/400/100/Batch-Fenster
```

Außerdem kann hier zusätzlich noch eine Zeit vorgegeben werden, die verstreichen soll, bevor das Fenster nach Abarbeitung des Batch-Files wieder verschwindet. Beispiel:

```
TOOL TYPE  DELAY=1000
```

Die Verzögerungszeit muß dabei in 1/50 Sekunden angegeben werden.

4.2 Neue Devices der Workbench-Version 1.3

Mit Hilfe des Mount-Befehls (s. Kapitel 2) lassen sich einige neue Geräte in das AmigaDOS einbinden. In den nun folgenden Abschnitten stellen wir diese Geräte nacheinander vor, geben Hinweise, was man bei der Einbindung der Geräte beachten muß und wie man mit ihnen umzugehen hat.

4.2.1 Das NewCon-Device

Mit diesem neuen Device haben Sie sehr wahrscheinlich schon Kontakt gehabt, ohne es zu wissen: Die Shell benutzt diese neue Fensterschnittstelle für die anfallenden Ein- und Ausgaben. Das NewCon-Device wird genauso angesprochen wie das alt bewährte Con-Device, das vom CLI verwendet wird. Vor der ersten Benutzung muß es, wie auch alle anderen neuen Geräte, mit Hilfe des Mount-Befehls in das System eingebunden werden (*mount NEWCON:*). Die hierfür notwendige Eintragung in der Datei Mountlist - diese befindet sich in dem DEVS-Ordner der Workbench-Diskette - sollte folgendermaßen aussehen:

```
NEWCON:   Handler = L:Newcon-Handler
          Priority = 5
          StackSize = 1000
#
```

Bis auf die Ediermöglichkeiten, die schon beim NewShell-Befehl ausführlich besprochen worden sind, verhält sich das NewCon-Device wie ein Con-Device (s. Kapitel 3).

4.2.2 Das RAD-Device

Die Abkürzung RAD steht für Recoverable Ram Disk. Bei diesem neuen Device handelt es sich um eine resetfeste RAM-Disk für den Amiga. Bei der herkömmlichen RAM-Disk mit dem Namen RAM sind die in ihr enthaltenen Daten nach jedem Neustart des Rechners verloren. Einen normalen Reset überstehen die Daten der RAD jedoch anstandslos. In den meisten Fällen überleben die Daten sogar eine vorangegangene Guru Meditation.

Dieser entscheidende Vorteil gegenüber der bekannten RAM-Disk muß, wie sollte es auch anders sein, mit einem mindestens ebenso entscheidenden Nachteil erkauft werden: Die RAD verwaltet aus Sicherheitsgründen den Speicherhaushalt nicht dynamisch. Es spielt daher keine Rolle, ob in ihr Daten gespeichert werden oder nicht: Ihr Bedarf an Systemspeicher richtet sich immer nach der Gesamtkapazität. Diese Gesamtkapazität wird vor der ersten Benutzung in der Mount-List festgelegt. Werden hier unüberlegte Eintragungen vorgenommen, kann es sehr leicht passieren, daß von 1 MByte Systemspeicher nach dem Mount-Befehl z.B. nur noch 200 KByte übrigbleiben. Eine typische Eintragung für das RAD-Device in der Mount-List sieht folgendermaßen aus:

```
RAD:        Device = ramdrive.device
            Unit   = 0
            Flags  = 0
            Surfaces  = 2
            BlocksPerTrack = 11
            Reserved = 2
            Interleave = 0
            LowCyl = 0
            HighCyl = 21
            Buffers = 5
            BufMemType = 1
#
```

Bevor die RAD durch den Befehl *mount rad:* eingebunden wird, muß ihre Kapazität durch die Angabe hinter HighCyl festgelegt werden. Jeder Cylinder besitzt dabei eine Speicherkapazität von 11 KByte. Würde die RAD mit der obigen Eintragung eingebun-

den, so hätte die Disk eine Gesamtkapazität von (21+1) * 11 KByte = 242 KByte.

Die RAD muß, wie jede andere Disk auch, vor ihrer ersten Benutzung mit Hilfe des Format-Befehls formatiert werden. Ein Beispiel:

```
format drive rad: name Test_Disk
```

Anschließend steht einem die resetfeste RAM-Disk zur Verfügung. Nach einem Reset genügt die Eingabe von *mount rad:*, um die RAD samt Inhalt wiederherzustellen. Sollte sich dabei herausstellen, daß Daten verlorengegangen sind, so kann man mit Hilfe des DiskDoctors (s. Kapitel 2) versuchen, die RAD zu restaurieren.

Wird die RAD nicht mehr benötigt, so kann der größte Teil des von ihr belegten Systemspeichers durch den Befehl Remrad (s. Kapitel 3) wieder freigegeben werden. Außerdem sollte bei dieser Gelegenheit der Mount-Befehl mit Hilfe des Assign-Option *unmount* rückgängig gemacht werden:

```
assign rad: unmount
```

Der gesamte Speicherbereich, den die RAD belegt, wird aber erst beim nächsten Boot-Vorgang wieder freigegeben.

4.2.3 Das Pipe-Device

Eine Pipe stellt einen Kommunikationskanal für den Datenaustausch zwischen verschiedenen Prozessen dar. Ein solcher Kommunikationskanal besteht im wesentlichen aus einem 4 KByte großen Datenpuffer, auf den von einem Prozeß aus schreibend und gleichzeitig von einem anderen Prozeß aus lesend zugegriffen werden kann. Damit ist es möglich, Daten ohne den Umweg über eine Hilfsdatei von Programm zu Programm zu übertragen.

Bevor eine Pipe benutzt werden kann, muß das Device gemountet werden (mount pipe:). Die hierfür notwendige Eintragung ist in der Mount-List der Workbench-Diskette vorhanden.

Es können theoretisch beliebig viele Kommunikationskanäle gleichzeitig geöffnet werden. Aus diesem Grunde kann der Device-Name Pipe: bei Bedarf noch um einen Kanalnamen ergänzt werden. In dem nun folgenden Beispiel wird das aktuelle Inhaltsverzeichnis über *pipe "test"* in den ED geladen.

```
dir >pipe:test
ed pipe:test
```

Früher wäre dies nur unter Zuhilfenahme einer Zwischendatei möglich gewesen:

```
dir >df0:hilfsdatei
ed df0:hilfsdatei
```

Sollte die Kapazität des Kanalpuffers nicht ausreichen, so wartet der Ausgabeprozeß so lange, bis von einem anderen Prozeß aus die Daten ausgelesen werden. Leitet man zum Beispiel die Ausgabe des gesamten Inhaltsverzeichnisses der Workbench-Diskette auf die Pipe um (*dir >pipe:test opt a*), so bleibt der Prozeß nach einer Weile hängen, da der Puffer keine weiteren Zeichen mehr aufnehmen kann. In dieser Situation kann nur noch eine zweite Shell helfen, von der aus die Pipe ausgelesen werden kann.

4.2.4 Das Speak-Device

Die Sprachausgabe des Amiga läßt sich mit Hilfe des Speak-Handlers der Workbench 1.3 ansprechen wie jedes andere Gerät auch. Vorher muß der Handler jedoch noch durch *mount speak:* in das System eingebunden werden.

Beispiele für Sprachausgaben:

```
echo >speak: "Nice to see you"
dir >speak df0: opt a
type s:startup-sequence to speak:
```

Theoretisch soll es möglich sein, mit Hilfe des Zusatzes OPT verschiedene Ausgabemodi auszuwählen. Der derzeitige Speak-Handler wertet diese Angaben jedoch noch nicht korrekt aus.

4.2.5 Das Aux-Device

Dieser Handler unterstützt die serielle Schnittstelle des Amiga. Im Gegensatz zu dem bekannten SER-Device werden die Daten aber nicht mehr zwischengespeichert. Hat man das Aux-Device "gemountet" (*mount aux:*), kann hiermit zum Beipiel eine Art Multiuser-Betrieb mit dem Amiga realisiert werden (s. Kapitel 5). Die Übertragungsparameter werden mit dem Preferences-Programm eingestellt.

4.2.6 Das Fast-File-System

Im Verzeichnis L der neuen Workbench-Diskette findet man außer den neuen Handler-Programmen auch eine Eintragung mit dem Namen FastFileSystem.

Allgemein kann ein File-System als ein um einige Funktionen erweiterter Handler angesehen werden. Es ist unter anderem direkt für die Organisation der Daten auf den Laufwerken verantwortlich. Auch ein File-System greift dabei nicht selbst auf die Hardware zu, sondern bedient sich eines Device-Treibers.

So wie der Speak-Handler das Speak-Device benutzt, so kann ein File-System zum Beispiel das Trackdisk-Device ansprechen, um Daten von einer Floppy-Station zu lesen. Ein File-System ist im Gegensatz zu den bisher besprochenen Handlern auf kein spezielles Device fixiert. Um eine Harddisk anzusprechen, genügt eine entsprechende Eintragung in der Mount-List - auf sie kommen wir noch zu sprechen - und dasselbe File-System greift auf ein Harddisk-Device zu.

Bisher lief die Kommunikation des Amiga mit den angeschlossenen Laufwerken (Floppy oder Festplatte) in jedem Fall über das File-System, welches sich fest im Betriebssystem Kickstart befindet. Speziell für Laufwerke, deren Speichermedium sich nicht beliebig auswechseln läßt (Festplatte, RAD-Disk) hat man nun das neue Fast File System (kurz: FFS) entwickelt. Der Name deutet schon auf die wesentliche Verbesserung an diesem File-System hin: Es ist um einiges schneller als das alte File-System.

Da ein Diskettenwechsel nicht ausgewertet wird, funktioniert das FFS nur in Verbindung mit einer Festplatte oder der neuen RAM-Disk RAD einwandfrei. Speziell bei Festplatten können wiederum auch nur die Partitions unter FFS laufen, die nicht automatisch gemountet werden, für die also eine Eintragung in der Mount-List existiert.

Um in den Genuß des neuen FFS zu gelangen, müssen in der Eintragung für das betreffende Gerät bzw. die betreffende Partition in der Mount-List folgende Zeilen hinzugefügt werden:

```
GlobVec = -1
FileSystem = L:FastFileSystem
DosType = 0x444F5301
```

Diese Änderungen können sehr leicht mit dem Editor ED ausgeführt werden. Der Aufruf des ED sieht dafür folgendermaßen aus:

```
ed devs:mountlist
```

Anschließend kann das Gerät durch *mount <Gerätename>* eingebunden werden. Diesen Befehl sollte man in der Startup-Sequence hinter einen eventuell vorhandenen BindDrivers-Befehl ansiedeln.

Vor dem ersten Einsatz muß das Laufwerk nun noch neu formatiert werden (dies gilt auch für die RAM-Disk RAD). Der Format-Befehl muß hierzu um den Zusatz FFS ergänzt werden. Beispiel:

```
format drive dh1: name Part_1 FFS
```

Soll bei einer Festplatte eine schon vorhandene Partitionierung beibehalten werden, so genügt es, nur jeweils die mit dem FFS versehenen Partitions neu zu formatieren. Falls auch nur bei einer Partition in der Mount-List die Angabe hinter LowCyl bzw. HighCyl geändert wird, so muß die gesamte Harddisk neu formatiert werden.

Besondere Beachtung gebührt bei der Verwendung des FFS dem Befehl AddBuffers (s. Kapitel 2). Anders als bei dem alten File-System resultiert aus einer Erhöhung des Pufferspeichers durch AddBuffers in jedem Fall eine Geschwindigkeitssteigerung. Besonders deutlich wird diese Steigerung bei der Ausgabe des Inhaltsverzeichnisses mit Hilfe des Dir-Befehles.

5. Hinweise zur Arbeit mit dem CLI

Aufgabe dieses Kapitels soll es sein, Probleme aus dem Weg zu räumen, mit denen jeder früher oder später bei der Arbeit im CLI konfrontiert wird. Außerdem werden hier jede Menge zusätzliche Informationen und Hinweise rund um das CLI gegeben.

5.1 Über Ein- und Ausgaben im CLI

Daß man CLI-Ausgaben durch Drücken einer beliebigen Taste kurzzeitig unterbinden kann, wissen Sie wahrscheinlich schon längst. Gerade beim Durchforsten längerer Directorys kommt man ohne diese Möglichkeit gar nicht aus. Was bewirkt aber nun genau ein Fortsetzen der Ausgabe? "Ein Drücken der Return- oder Enter-Taste", werden Sie nun sagen. Das ist aber nur die halbe Wahrheit, denn es kommt nur darauf an, daß keine Tastaturanforderungen an das CLI mehr vorliegen. Das ist zwar nach dem Drücken dieser Tasten genau der Fall, aber es gibt noch eine weitere Möglichkeit, die Ausgabe fortzusetzen, nämlich mittels der Backspace-Taste (Pfeil nach links). Werden durch sie sämtliche eingegebenen Zeichen wieder gelöscht, so hat dies die gleiche Wirkung wie die Betätigung der obengenannten Datenübernahme-Tasten.

Dieses Unterbinden der Ausgabe hat aber noch eine ganz andere Aufgabe: Es ist möglich, einen neuen Befehl einzugeben, noch während ein alter bearbeitet wird. Damit nun nicht alle Ein- und Ausgaben durcheinanderlaufen, werden für die Zeit der Eingabe des neuen Kommandos alle Ausgaben gesperrt. Das hat natürlich noch gar nichts mit den Multitasking-Fähigkeiten des Amiga zu tun, denn alle eingegebenen Befehle werden immer noch schön hintereinander ausgeführt. Trotzdem kann man hierdurch etwas schneller arbeiten, da ja das DOS nun durchgehend aktiv ist. Bei manchen Befehlssequenzen sollte man aber auf diese Eingabemöglichkeit verzichten. Dazu ein Beispiel:

```
copy text to kopie
delete text
```

Sollte der Kopiervorgang aus irgendeinem Grund scheitern (z.B. weil die Diskette voll ist), wird anschließend ohne Hemmungen noch die Datei Text gelöscht. Nun haben Sie hiervon weder eine Kopie noch das Original. Ein Tip: Der DiskDoctor, ein normaler CLI-Befehl, kann in solchen Fällen noch einmal helfen. Voraussetzung hierfür ist jedoch, daß noch kein Schreibzugriff auf die von der Datei ehemals belegten Blocks stattgefunden hat. Ausprobieren sollte man es in jedem Fall.

5.2 Die Jokerzeichen

Alte Commodore-Hasen werden die Jokerzeichen * bzw. ? auf dem Amiga schon vermißt haben. Sie ermöglichen auf Rechnern der älteren Generation (z.B. PET, VC20, C64) eine verkürzte Eingabe von Dateinamen. Will man zum Beispiel seine Programme Test1, Test2 bis Testn von der Diskette löschen, setzt man in den Scratch-(=Lösch-)Befehl einfach den Namen *Test** ein. Automatisch werden hierdurch alle Files, die mit Test beginnen (und irgendwie enden), von der Diskette gelöscht. Will man aber gezielt nur die Dateien ansprechen, deren File-Name genau 5 Buchstaben lang ist, kann man dazu das Fragezeichen verwenden: *Test?*. Eine Datei namens Test1Version würde in diesem Fall nicht gelöscht werden. Zusätzlich sind auf diesen Computerveteranen auch Kombinationen der beiden Joker zugelassen, wie:

```
"????Version*"
```

In diesem Fall würden alle Files angesprochen, die ab der fünften Position im File-Namen Version stehen haben und die beliebig enden. Beispielsweise erfüllen die folgenden Dateinamen diese Bedingungen:

```
TestVersion1
EineVersion
LastVersionOfToday
```

Damit dürfte nun auch der Computerneuling etwas mit dem Begriff des Jokers anfangen können. Und nun die gute Nachricht: Solche Spaßmacher gibt's auch auf dem Amiga, der eine ist auch hier wieder das Fragezeichen, als Äquivalent zum Sternchen

kann in etwa das "Lattenkreuz" (#) angesehen werden. Das Fragezeichen übernimmt wieder die No-Name-Funktion für genau eine Zeichenposition innerhalb des File-Namens. Das Symbol # ist jedoch um einiges flexibler als das Sternchen. Wie Sie vielleicht wissen, bedeutet ein vorangestelltes # im englischsprachigen Raum meist nichts anderes als das Wort Number. Es wird also (unter anderem) noch ein Zahlenwert hinter dem Symbol erwartet. Erweitert man z.B. den Befehl *Dir* um die Zeichenkombination *Test#3a*, so werden nur File-Namen gesucht, die mit drei A's enden. Diese Funktion bildet natürlich in keiner Weise das Sternchen nach. Dafür wartet das Doppelkreuz aber noch mit einer weiteren Besonderheit auf: Gibt man anstelle der Zahl direkt ein Fragezeichen ein, so werden alle folgenden Zeichen im File-Namen ignoriert. Hier haben wir es also mit dem echten *-Ersatz zu tun. Dabei besteht zusätzlich die Möglichkeit, hinter der #?-Kombination den weiteren Verlauf des File-Namens anzugeben. In diesem Fall kann zum Beispiel nach allen Files gesucht werden, die mit .info enden. Die Eingabe muß dann *#?.info* lauten.

Nach soviel Theorie nun erst einmal ein paar Beispiele zum Mitmachen. Ihre Aufgabe soll es sein, herauszufinden, welche der acht angegebenen File-Namen jeweils ausgegeben würden, falls man den Dir-Befehl um die unter 1 bis 7 angegebenen Parameter erweiterte.

Liste der vorhandenen File-Namen:

Katze.1 1.Hund
Katze.2 2.Hund
Katzenjammer Hundebellen
Miezekatzen krummer_Hund

Liste der Dir-Anweisungen:

1. Katze.?
2. K#?
3. ???????????
4. #?
5. Hunde?????
6. #?Hund
7. ????????#2m#? (doch, das gibt's !)

Nun, alles gewußt? Dann vergleichen Sie doch vorsichtshalber einmal Ihre Ergebnisse mit der Lösung:

```
zu 1: Katze.1, Katze.2
zu 2: Katze.1, Katze.2, Katzenjammer, krummer_Hund
zu 3: Hundebellen, Miezekatzen
zu 4: (alle Files)
zu 5: (kein File;)
zu 6: Hund, 2.Hund, krummer_Hund
zu 7: Katzenjammer
```

Das letzte Beispiel zeigt sehr deutlich, daß auch Kombinationen möglich sind, deren Sinn zumindest recht fragwürdig ist.

5.3 Unterbrechungen im CLI

Abbruch über die Tastatur

Eine Run-/Stop-Taste, wie sie vielleicht noch einige Besitzer eines C64 kennen, findet man auf der Tastatur des Amiga nicht. Aber natürlich ermöglicht auch dieser Rechner die Unterbrechung eines gerade ablaufenden CLI-Befehls. Hierfür ist die Tastenkombination Ctrl und C vorgesehen. Alle CLI-Befehle reagieren darauf mit der Rückkehr zu dem Programm, von dem aus sie aufgerufen wurden. In den meisten Fällen wird dies das CLI sein. Als Bestätigung der vorzeitigen Beendigung des Befehls erscheint dort die Meldung "*** Break". Die drei Sternchen geben an, daß eine Unterbrechung vom Typ C stattgefunden hat. Außer dieser stärksten Form der Unterbrechungsanforderung gibt es noch die Kommandos Ctrl und D bis Ctrl und F, deren Auftreten durch weniger Sternchen geschmückt wird. Inwieweit diese Befehle jedoch überhaupt ausgewertet werden, hängt von den jeweiligen CLI-Befehlen ab, auf die sie angewendet werden. Bei den derzeit vorhandenen Befehlen ist nur noch Ctrl und D interessant, und zwar in Verbindung mit Search und Execute.

Der Search-Befehl, mit dem in ganzen Verzeichnissen nach dem Auftreten einer vorgegebenen Zeichenkette gesucht werden kann, reagiert auf Ctrl und C wie jeder andere Befehl auch: Der Vorgang wird sofort abgebrochen. Gibt man jedoch Ctrl und D

ein, wird das aktuelle File, in dem gesucht wird, verlassen und sofort in der nächsten Datei weitergesucht. Ctrl und D ist also in seiner Wirkung schwächer als Ctrl und C.

Etwas heikel wird die Angelegenheit beim Execute-Befehl. Er dient zur Ausführung sogenannter Batch-Dateien, die wiederum aus CLI-Befehlen bestehen. Sollten Sie mit dieser Erklärung noch nicht viel anfangen können, finden Sie im gesamten Kapitel 6 genügend Informationen zu diesem Thema.

Grundsätzlich hat auch bei der Ausführung eines CLI-Befehls innerhalb einer Batch-Datei Ctrl und C immer die höhere Priorität als Ctrl und D. Der laufende CLI-Befehl wird hierduch sofort beendet, und es wird auf die Execute-Ebene zurückgekehrt. Es liegt dabei an dem unterbrochenen Befehl, was nun geschieht: Liefert er aufgrund der Unterbrechung einen Errorcode an Execute zurück (Fehlernummer >= 10), so wird auch die Batch-Datei unter Ausgabe einer entsprechenden Meldung sofort verlassen (Beispiel: Ctrl und C bei Ausführung des Search-Befehls). Sollte der Befehl aber ohne eine Fehlermeldung auf die Execute-Ebene zurückkehren, wird die Batch-Datei ganz normal weiter abgearbeitet (Beispiel: Ctrl und C bei Ausführung des Dir-Befehls).

Eine besondere Funktion hat bei dem Befehl Execute die Eingabe von Ctrl und D. Bei Eingabe dieser Tastenkombination wird der gerade aktive CLI-Befehl der Batch-Datei noch vollständig abgearbeitet und anschließend die Datei in jedem Fall verlassen. Problematisch wird die Angelegenheit, wenn der aktive CLI-Befehl selber schon auf ein Ctrl und D reagiert:

Der Search-Befehl wertet ein Ctrl und D aus (s.o.). Gibt man nun während der Abarbeitung dieses Kommandos durch eine Batch-Datei das Kommando Ctrl und D, so reagiert der Search-Befehl hierauf und nicht etwa das Execute. Nach Beendigung des Suchvorganges wird also die Batch-Datei nicht verlassen, es sei denn, es folgen sowieso keine Befehle mehr.

Abbruch per Befehl

Haben Sie sich schon einmal Gedanken darüber gemacht, wie man einen Befehl abbrechen könnte, der gar kein eigenes Eingabe-Fenster besitzt?

Wird ein Befehl durch Run gestartet, wird ein eigenständiger Prozeß erzeugt, an den vom ursprünglichen CLI-Fenster aus, keine Eingaben mehr geschickt werden können. Es handelt sich hierbei um einen sogenannten "nicht-interaktiven Prozeß". Lediglich die Ausgaben erscheinen auf dem CLI-Fenster. Ein einfaches Beispiel:

```
run dir df0:
```

Nach Ausgabe der neuen Nummer, die für diesen Prozeß vergeben wurde (z.B. *[CLI 2]*), wird das Inhaltsverzeichnis der Diskette im aktuellen CLI-Fenster angezeigt. Auf ein Ctrl und C reagiert der Befehl nicht mehr. Die Ausgabe wird ungestört fortgesetzt.

Dennoch gibt es eine Möglichkeit, einen solchen nicht-interaktiven Prozeß zu verlassen. Der CLI-Befehl *Break* drückt sozusagen softwaremäßig auf die Ctrl-Taste eines Prozesses. Der Befehl erwartet dazu die Nummer des Prozesses, der den Befehl erhalten soll, sowie den entsprechenden Buchstaben für die Art der Unterbrechung (c-f). Sollen alle Unterbrechungsanforderungen geschickt werden, kann hierfür auch *all* eingesetzt werden.

Für die Unterbrechung der Directory-Ausgabe aus dem obigen Beispiel müßte der Befehl also lauten:

```
break 2 c
```

Man darf dabei keine Hemmungen haben, in das gerade mit der Ausgabe beschäftigte Fenster zu schreiben. Sobald ein Zeichen eingegeben wird, wartet die Ausgabe des Inhaltsverzeichnisses bis zur Eingabe des Return-Zeichens.

Der Break-Befehl kann natürlich auch von einem CLI aus die Abarbeitung eines Befehls, der in einem anderen CLI eingegeben wurde, unterbrechen. Man muß nur die entsprechende Prozeß-

nummer eingeben, die ja hier mit der CLI-Nummer übereinstimmt.

5.4 Die RAM-Disk und das CLI

Dieser Abschnitt richtet sich in erster Linie an die Besitzer nur eines Floppylaufwerkes. Bei dieser Minimalkonfiguration tauchen häufig Schwierigkeiten auf, die mit Hilfe der RAM-Disk beseitigt werden können. Die glücklichen Besitzer mehrerer Floppys kommen aber auch nicht zu kurz: Sie können den folgenden Ausführungen entnehmen, wann sich bei ihnen der Einsatz der RAM-Disk lohnt.

CLI-Befehle in der RAM-Disk

Den ersten CLI-Frust erlebt man meist, wenn man die Workbench-Diskette aus dem Laufwerk gegen eine beliebige andere austauscht und nun versucht, sich mittels *Dir* deren Inhaltsverzeichnis anzusehen. Sofort erscheint oben links auf dem Bildschirm ein dicker Requester mit der Aufforderung, doch bitteschön wieder die Workbench-Diskette in ein (also: in das) Laufwerk zu legen. Kommt man der Aufforderung nicht nach und klickt einfach das Feld Cancel an, bekommt man unverblümt mitgeteilt, daß es einen Befehl Dir nicht gebe. Gut, dann soll er die Diskette eben haben. Nun kommt die zweite Überraschung: Es wird nicht etwa das gewünschte Inhaltsverzeichnis ausgegeben, sondern das der Workbench-Diskette. Aber man ist ja nicht einfaltslos: Beim nächsten Versuch gibt man ein *Dir df0:*, damit der Rechner weiß, welche Diskette gemeint ist. Ergebnis: Keine Veränderung. Als CLI-Neuling ist man nun mit seinem Latein am Ende und will aufgeben, es sei denn, man hat Zugang zu den folgenden Informationen.

Wie schon im Kapitel 1 bei der Einführung erwähnt wurde, ist jeder CLI-Befehl ja nichts anderes als ein kleines Programm, das vor der Ausführung jedesmal erst von der Diskette geladen werden muß. Im Normalfall befinden sich diese Befehle jedoch nur auf der Diskette, von der das System gestartet wurde. Auf der Workbench-Diskette sind die Befehle im Verzeichnis C unterge-

bracht. Das Betriebssystem des Amiga merkt sich beim Start genau, welches die Systemdiskette ist - sie kann jederzeit unter dem Namen Sys: angesprochen werden - und in welchem Verzeichnis dieser Diskette die CLI-Befehle stehen.

Gibt man nun einen solchen Befehl ein, wird sofort genau diese Diskette verlangt, falls sie nicht schon eingelegt ist.

In unserem obigen Beispiel haben wir nur den Fehler gemacht, dem Dir-Befehl nicht den richtigen Namen der Diskette mitzugeben, deren Inhaltsverzeichnis wir ansehen möchten. Denn df0: ist nur die Bezeichnung des Laufwerkes. Findet das DOS den Dir-Befehl auf der Workbench-Diskette, wird nun nicht mehr die von uns gewünschte Diskette verlangt, sondern eben das Inhaltsverzeichnis der Diskette ausgegeben, die sich zu diesem Zeitpunkt im Laufwerk Null befindet.

Mit dem Befehl *Dir Spiele:* zum Beispiel wäre das nicht passiert. Das DOS würde in dem Fall zuerst nach der Workbench-Diskette fragen, von dieser den Befehl Dir laden, anschließend nach einer Diskette mit dem Namen Spiele verlangen und nun erst deren Inhaltsverzeichnis ausgeben.

Jetzt nur nicht nervös werden, es geht auch einfacher. Legen Sie einfach die Workbench-Diskette ein, und tippen Sie die folgenden drei Zeilen ab:

```
makedir ram:c
copy from df0:c/dir to ram:c
path ram:c add
```

Anschließend können Sie jede beliebige Diskette einlegen und sich sofort deren Inhaltsverzeichnis durch Dir df0: ansehen. Der Dir-Befehl wird hierbei innerhalb kürzester Zeit aus der RAM-Disk geladen. Die einzelnen Befehle haben folgende Funktionen:

Zeile 1

Es wird ein beliebiges Unterverzeichnis (hier C) in der RAM-Disk erzeugt.

Zeile 2

Der Befehl Dir wird von der eingelegten Diskette in dieses Unterverzeichnis kopiert. Dahinter könnten noch weitere Kopieranweisungen für andere CLI-Befehle folgen.

Zeile 3

Dem DOS wird mitgeteilt, daß vor der Frage nach der Diskette mit den CLI-Befehlen erst noch in der RAM-Disk im Unterverzeichnis C nachgesehen werden soll. Nur wenn der Befehl hier nicht zu finden ist, soll nach der Diskette verlangt werden.

Außer durch den Befehl Path gibt es noch eine zweite Möglichkeit, wie man DOS dazu bewegen kann, CLI-Befehle aus der RAM-Disk zu holen. Der Befehl Assign ermöglicht es, von vornherein einen vollständig neuen Pfadnamen für die Suche nach allen CLI-Befehlen vorzugeben. Schauen Sie sich zunächst einmal die Liste an, die der Amiga ausgibt, wenn man nur Assign eingibt. Dort finden Sie unter der Überschrift Directorys: auch einen Eintrag C. Rechts daneben ist diesem C ein Pfadname zugeordnet. Dort steht zum Beispiel: A500 WB1.2 D:c. Das AmigaDOS sucht nun immer in dem Verzeichnis nach den CLI-Befehlen, die hier angegeben sind. Beim Start des Rechners wird hier automatisch das Verzeichnis C der System-Diskette eingetragen. Durch den Befehl Assign kann diese Eintragung sehr leicht verändert werden:

```
makedir RAM:c
assign C: RAM:c
```

Ihr Amiga wird nun nur die Befehle verstehen, die sich in der RAM-Disk im Unterverzeichnis C befinden. Man sollte also direkt hinter MakeDir die nötigen Befehle in dieses Unterverzeichnis kopieren. Um dennoch an die CLI-Befehle der eingelegten Systemdiskette zu kommen, muß man jetzt ihren vollständigen Pfadnamen angeben:

```
sys:c/assign C: sys:c
```

Hierdurch wird die Zuweisung der RAM-Disk wieder rückgängig gemacht, damit Sie in aller Ruhe erst einmal ein paar Be-

fehle kopieren können. Durch *Sys:* wird automatisch die Systemdiskette angesprochen. Liegt sie nicht im Laufwerk, wird sie in einem Requester verlangt.

Ein Grundgerüst an CLI-Befehlen erzielt man durch die Eingabe der folgenden Zeilen:

```
makedir ram:c   ;(falls noch nicht geschehen !)
copy c/assign  ram:c
copy c/dir     ram:c
copy c/cd      ram:c
copy c/copy    ram:c
copy c/delete  ram:c
copy c/makedir ram:c
copy c/rename  ram:c
copy c/newcli  ram:c
copy c/endcli  ram:c
copy c/run     ram:c
copy c/list    ram:c
assign C: ram:c
```

Natürlich können Sie noch weitere CLI-Befehle in die RAM-Disk kopieren. Man sollte sich aber auf die wichtigsten beschränken, da ein Befehl wie SetClock oder Date dort wirklich nur Speicher verschwendet. Besitzer eines Amiga mit "nur" 500 KByte Speicherkapazität können hierbei sogar in ernsthafte Schwierigkeiten kommen.

Sollte wirklich einmal ein Befehl benötigt werden, der nicht in der RAM-Disk vorhanden ist, können Sie entweder mit Assign wieder auf die Systemdiskette zurückschalten (*assign C: sys:c*) oder durch Angabe des vollständigen Pfadnamens auf den fehlenden Befehl zugreifen: *sys:c/date 29-oct-87 11:30:35*.

Wenn Ihnen die Eingabe der obigen Befehlsliste vor jeder Arbeit mit dem CLI zu langwierig erscheint, können Sie die Prozedur auch automatisieren. Im Kapitel 4 gibt es jede Menge Hinweise hierzu.

Weitergehende Informationen zu den Befehlen MakeDir, Copy, Path und Assign finden Sie übrigens im Kapitel 2.

Kopieren mit der RAM-Disk

Mit zwei Laufwerken ist das Kopieren von einzelnen Dateien oder ganzen Ordnern kein Problem. Der Kopierbefehl könnte zum Beispiel lauten:

```
copy df0:utilities/notepad df1:Hilfsprogramme
```

Wie soll man aber mit nur einem Laufwerk Kopien anfertigen? Eine Möglichkeit besteht darin, anstelle der Laufwerksnummern die Namen der Disketten anzugeben. Zum Beispiel:

```
copy from Textomat:Briefe/Peter to Texte:Briefe
```

DOS fragt nun automatisch abwechselnd nach diesen Disketten und orientiert sich nicht mehr an den Laufwerksnummern. Diese Methode hat aber zwei große Nachteile:

1. Man muß immer die Namen der beiden Disketten wissen.
2. Selbst bei einer kleinen Datei von nur einigen Bytes muß mindestens 7 (in Worten: sieben) mal die Diskette gewechselt werden.

Dieses Verfahren dürfte daher für eine ernsthafte Anwendung ausscheiden. Die bessere Kopiermethode benutzt die RAM-Disk zur Zwischenspeicherung der Dateien. Die Vorgehensweise ist dabei ganz einfach: Zuerst wird die Diskette eingelegt, von der etwas kopiert werden soll. Die entsprechenden Dateien werden durch den Copy-Befehl in die RAM-Disk gebracht.

Anschließend wird die Zieldiskette in das Laufwerk gelegt, und die gewünschten Dateien werden von der RAM-Disk hierauf kopiert. Beispiel: Es soll eine Kopie des gesamten C-Verzeichnisses auf einer anderen Diskette angelegt werden:

1. Quelldiskette einlegen.
2. Der Ordnerinhalt wird ins RAM kopiert:

   ```
   copy df0:c ram:
   ```

3. Zieldiskette einlegen.

4. Ein Ordner namens "c" wird erzeugt:

```
makedir df0:c
```

5. Der Ordnerinhalt wird auf die Zieldiskette kopiert:

```
copy ram: df0:c
```

Das Ganze kann natürlich nur funktionieren, wenn man vorher den Copy- und MakeDir-Befehl in die RAM-Disk kopiert und dies dem System mit Assign oder Path mitteilt. Die beiden Befehle müssen nämlich auch dann vorhanden sein, wenn sich die Systemdiskette nicht gerade im Laufwerk befindet.

Nach dem Kopiervorgang sollten die Dateien in der RAM-Disk wieder gelöscht werden, um den von ihnen belegten Speicherplatz wieder dem System zuzuführen. Der Befehl hierzu lautet:

```
delete ram:c all.
```

Hierdurch wird das gesamte Verzeichnis C, das für den obigen Kopiervorgang in der RAM-Disk angelegt wurde, wieder gelöscht.

5.5 Drucken vom CLI aus

Die von der Workbench stiefmütterlich behandelten Druckerfreunde sollen im folgenden auf ihre Kosten kommen. Beschäftigen sich die ersten Abschnitte noch mit den Grundlagen des Druckens im CLI, geht es anschließend um die Bewältigung der Probleme, die auch den CLI-Profi zur Verzweiflung treiben können.

5.5.1 File-Ausdruck mit Copy

Der Befehl Copy ist Ihnen sicherlich schon öfter begegnet. Zur Erinnerung: Er ermöglicht das Duplizieren von Dateien bzw. ganzen Ordnern. Die Syntax lautet:

```
COPY FROM,TO/A,ALL/S,QUIET/S
```

From und To geben die Quelle und das Ziel des Vorgangs an; durch den Zusatz *all* werden alle Files eines From-Directorys angesprochen. *Quiet* schließlich bewirkt eine "leise" Bearbeitung des Befehls, was nicht ganz wörtlich zu nehmen ist: Es werden dabei einfach nur die sonst auftretenden Ausgaben unterdrückt.

Das Duplizieren hat es nun aber in sich, denn es können alle an Ihren Amiga angeschlossenen Geräte angesprochen werden, und zwar nicht nur Floppy oder Harddisk. Die möglichen Datenflußrichtungen hängen dabei vom jeweiligen Gerät ab. Der Drucker als Datenquelle ist natürlich Unsinn, er kann also nur ein Zielgerät sein, während z.B. eine Floppy beide Datenrichtungen (Laden und Speichern) erlaubt.

Eine Liste der Geräte, die Ihr Amiga kennt, erhalten Sie nach der Eingabe des Befehls Assign. Je nach Konfiguration stehen dort unter Devices z.B.:

```
DF0 DF1 PRT PAR SER RAW CON RAM
```

Uns interessiert aber nur der Drucker, der auch in Ihrer Liste durch den Namen Prt für Printer vertreten sein wird.

Möchte man nun ein Textfile schwarz auf weiß besitzen, kann man durch *copy name_des_files to prt:* sehr schnell den gewünschten Ausdruck erzielen.

Welches Format der Ausdruck besitzt? Nun, das hängt von den gewählten Parametern im Programm Preferences ab. Sie kennen es sicher noch von der guten alten Workbench. Was dort eingestellt ist, zählt! Wenn also z.B. eine serielle Datenübertragung gewünscht wird, handelt es sich bei Prt um ein serielles Device, ansonsten um ein paralleles.

Das Drucker-Device wird also praktisch durch die Preferences "bevormundet". Ein Programm, das Daten zum Drucker schickt, muß sich keine Sorgen mehr um Schnittstellentypen, Druckbreiten oder Papierlängen machen. Die Steuerzeichen werden automatisch für den angewählten Drucker eingefügt, die Daten also erst noch interpretiert. Für spezielle Anwendungen möchte man dies jedoch umgehen. Dafür stehen die Roh-Treiber PAR

und SER zur Verfügung. Daten, die an sie übergeben werden, gehen direkt 'raus zur Peripherie.

5.5.2 Allgemeines Umlenken der Ausgabe

Mit dem Copy-Befehl haben wir die ersten Druckerhürden doch ganz gut genommen. Wie bekommt man aber nun eine Ausgabe, die ein Programm gerade auf dem Bildschirm erzeugt, in eine Datei, die man dann mittels *Copy* auf den Drucker ausgeben kann? Viel zu kompliziert gedacht! Es geht viel einfacher. Sämtliche Ausgaben der DOS-Befehle können auf ein beliebiges Gerät umgeleitet werden. Sinnvollerweise hat man als Default-Device (=voreingestelltes Gerät) den Bildschirm gewählt. Niemand wird aber daran gehindert, durch das Anhängen von *>SER:* an den Befehl *Dir* das Inhaltsverzeichnis seiner Workbench-Diskette via Modem nach Tokio zu schicken.

Im Ernst: Alle Devices, die Daten entgegennehmen können, also auch Ihr Drucker prt:, sind für die Ausgabe zugelassen. Einfach die spitze Klammer > hinter das Kommando setzen, gefolgt von dem Device-Namen (z.B. *PRT*), und ab geht's. Man muß bei der Umlenkung von Ausgaben jedoch auf die Reihenfolge eventueller Befehlsoptionen achten: Zuerst steht immer das *redirect the output-Kommando* (also unser >-Zeichen), gefolgt vom Device-Namen. Erst dann dürfen die zusätzlichen Angaben für den Befehl erscheinen. Dazu sind ein paar Beispiele angebracht:

```
dir >prt:
```

Gibt das aktuelle Inhaltsverzeichnis auf den Drucker aus.

```
dir >prt: opt a
```

Gibt alle Verzeichnisse der aktuellen Disk auf den Drucker aus.

```
dir >prt: ram: opt a
```

Gibt alle Verzeichnisse der RAM-Disk auf den Drucker aus.

```
echo >prt: "Hallo"
```

Der Texte "Hallo" wird ausgedruckt.

```
type >prt: df0:s/startup-sequence
```

Die Startup-Sequenz wird ausgedruckt.

5.5.3 Steuerzeichen für den Drucker

Das Programm Preferences auf der Worbench-Diskette wirkt auf den ersten Blick recht leistungsfähig. Wenn man dann aber einmal die versprochenen Sonderfunktionen der Druckerbetriebsanleitung mit den angebotenen Menüpunkten in den Preferences vergleicht, wird man doch meistens seitens des Amiga einige Dinge vermissen: Hier gibt es zum Beispiel keine Möglichkeiten, um verschiedene nationale Zeichensätze einzuschalten oder den Doppeldruckmodus zu aktivieren. Von AmigaBASIC aus stellt dies zumindest prinzipiell kein größeres Problem dar (siehe Seite 5-3 im AmigaBASIC-Handbuch). Die CHR$-Funktion, mit der von BASIC aus auch Steuerzeichen an die Peripherie geschickt werden können, gibt es aber im CLI nicht, so daß hier andere Wege beschritten werden müssen.

Das grundlegende Problem der Steuerzeichen ist, daß man sie auf der Tastatur nicht direkt finden kann. Einige von ihnen können aber durch das gleichzeitige Drücken von Ctrl zusammen mit einem anderen Zeichen erreicht werden. Für die meisten Epson-Drucker zum Beispiel bedeutet das Steuerzeichen 15 das Einschalten des Schmalschrift-Modus. Die zugehörige Ctrl-Kombination lautet: Ctrl und O. Dabei ist es kein Zufall, daß das O gerade der 15. Buchstabe des Alphabets ist. Analog kann man so aus jedem Steuerzeichen den entsprechenden Buchstaben "berechnen". Aber aufgepaßt: Das O ist wirklich ein O und kein o (Shift nicht vergessen!). Das einzig Unschöne an der Sache ist, daß man auf dem Bildschirm nichts sieht, wenn man ein solches Zeichen eingegeben hat. Aber das macht ja nichts: Hauptsache ist, der Drucker versteht es. So ohne weiteres kann er das natürlich auch nicht. Man muß die Ausgabe schon noch umleiten. Wie dies geschieht, wissen Sie (hoffentlich) noch aus dem letzten Abschnitt.

Am sinnvollsten wird die Umleitung auf den Befehl Echo angewandt. Er ist ja speziell vorgesehen für die reine Ausgabe von Texten. Das Ganze sieht dann folgendermaßen aus:

```
echo >prt: [Ctrl und O]
```

Die eckigen Klammern sollen andeuten, daß nicht der Text, sondern die angegebene Tastenkombination eingegeben werden muß. Das Leerzeichen vor dem Ctrl und O ist keine Zierde! Schließlich möchte man ja kein Unterverzeichnis im Drucker ansprechen. Bis auf Widerruf ([Ctrl und T]) wird nun alles in Schmalschrift ausgedruckt.

Schwierig wird es jedoch bei dem Kommando Ctrl und C, das ja, wie Sie wissen, auch dazu dient, einen CLI-Befehl abzubrechen. Dies Zeichen läßt sich ohne weiteres nicht übertragen. Hier hilft nur ein Trick:

Erzeugen Sie mittels BASIC eine Datei auf der Diskette, an die Sie als einziges Zeichen das Ctrl und C in Form eines CHR$(3) übergeben (A=1, B=2, C=3 !). Das folgende Mini-Programm erledigt diese Aufgabe für Sie:

```
OPEN "df0:CTRL-C" FOR OUTPUT AS #1
PRINT#1,CHR$(3);
CLOSE #1
```

Nun kann auch ein Ctrl und C gesendet werden, und zwar durch

```
COPY FROM df0:CTRL-C TO PRT:
```

oder durch (wir sind ja flexibel)

```
TYPE >prt: df0:CTRL-C
```

Schließlich gibt es noch eine Vielzahl sogenannter Escape-Sequenzen. Hierbei handelt es sich gleich um eine ganze Anzahl von Steuerzeichen, die durch das Sonderzeichen ESC eingeleitet werden und nacheinander an den Drucker geschickt werden müssen. Je nach Befehl folgt dem ESC noch mindestens ein Ctrl-Zeichen. Das ESC selber kann man zur Abwechslung mal wieder auf der Tastatur finden (ganz oben links). Als Beispiel dient ein typischer Epson-Befehl. Er würde in BASIC lauten:

```
PRINT#1,CHR$(27);"R";CHR$(9);
```

Durch das CHR$(9) wird der norwegische Zeichensatz aktiviert. Das Äquivalent im CLI sieht folgendermaßen aus:

```
echo >prt: [ESC] R [Ctrl I]
```

Hinter dem Doppelpunkt muß ein Leerzeichen kommen, vor und hinter dem R darf keines stehen.

Natürlich können auch hier Schwierigkeiten auftauchen: Gerade für Programmierer der Sprache C sind die geschweiften und eckigen Klammern {} bzw. [] sehr von Interesse. Diese könnten durch Einschalten des amerikanischen Zeichensatzes zum Vorschein gebracht werden, wenn es noch einen Buchstaben vor dem A im Alphabet gäbe. Man benötigt hierfür nicht CHR(9), wie bei dem obigen Beispiel, sondern ein CHR$(0), das es als Ctrl-Code auf der Tastatur so nicht gibt. Abhilfe leistet wieder ein BASIC-Programm, das die gewünschte Sequenz in einem File ablegt:

```
OPEN "df0:USA-Satz" FOR OUTPUT AS #1
PRINT#1,CHR$(27);"R";CHR$(0);
CLOSE #1
```

Das nach dem Start erzeugte File können Sie auf Ihre CLI-Arbeitsdiskette kopieren. Folgende Eingabezeile aktiviert dann den neuen Zeichensatz:

```
COPY USA-Satz prt:
```

bzw.

```
TYPE >prt: df0:USA-Satz
```

5.6 Anwendungen zum Con-Device

Mit dem Console-Device lassen sich im CLI einige nette Dinge anstellen. Die folgenden Abschnitte sollen Ihnen nur ein paar Anregungen für eigene Experimente mit diesem Device geben.

Eigene Fenster für CLI-Ausgaben

Grundsätzlich kann man alle Ausgaben, die von CLI-Befehlen auf dem aktuellen Fenster erzeugt werden, auch auf andere Geräte umleiten. Besonders nützlich ist es in einigen Fällen, die Ausgabe auf ein eigenes Fenster umzuleiten.

Möchte man zum Beispiel das Inhaltsverzeichnis einer Diskette ansehen, aber weiterhin in dem CLI weiterarbeiten, muß für diesen einen Befehl nicht extra ein neues CLI-Fenster mit NewCLI geöffnet werden. Der folgende Befehl gibt zum Beispiel das Inhaltsverzeichnis der Diskette in Laufwerk df0: in einem eigenen Fenster aus:

```
run dir >con:10/10/400/100/Inhalt df0:
```

Die Ausgabe kann jederzeit angehalten werden, indem man das Fenster anklickt und eine beliebige Taste drückt. Durch Return wird die Ausgabe fortgesetzt. Nach Beendigung des Befehls verschwindet das Fenster automatisch wieder.

Der Drucker als Schreibmaschine

Nichts ist vom CLI aus einfacher, als einen angeschlossenen Drucker zu einer komfortablen Schreibmaschine umzufunktionieren. Dazu genügt die Eingabe der folgenden Befehlszeile:

```
copy * to prt:
```

Mit dem Sternchen wird für die Eingaben ein ganz spezielles Con-Fenster angesprochen, das des jeweiligen CLI. Alle Eingaben werden nun nach dem Drücken der Return-Taste auf dem Drucker ausgegeben. Vorteil gegenüber einer normalen Schreibmaschine: Bevor die Zeichen abgeschickt werden, können mittels Backspace noch Korrekturen an der Zeile vorgenommen werden. Wie im CLI kann durch Ctrl und X sogar die gesamte Zeile wieder gelöscht werden. Zum CLI-Prompt zurück gelangt man durch die Eingabe von Ctrl und \.

Textdateien erstellen

Wenn man schnell einmal eine kleine Textdatei erzeugen will, bietet sich der folgende Befehl an:

```
copy * to df0:text
```

Der Unterschied zu dem Beispiel aus dem letzten Abschnitt besteht nur darin, daß die Zeichen hier nicht an den Drucker, sondern an das Diskettenlaufwerk geschickt werden. Dort werden sie in einer Datei mit dem Namen Text abgelegt. Zur Erstellung kleinerer Textdateien kann man sich dadurch häufig das Laden der relativ umfangreichen Programme ED bzw. Edit sparen. Auch diese Funktion kann durch die Eingabe von Ctrl und \ beendet werden.

Textdateien erweitern

Möchte man eine schon bestehende Textdatei noch um einen Textkopf (z.B. eine Programmbeschreibung) erweitern, kann dies sehr leicht durch den nun folgenden Befehl erreicht werden:

```
join con:10/10/400/100/Eingabe df0:textdatei as df0:neudatei
```

Die ursprüngliche Aufgabe des Join-Befehls besteht darin, mehrere Dateien zu einer einzigen zusammenzufügen. In unserem Fall besteht die erste Datei aus einem Con-Fenster, die zweite Datei ist der schon bestehende Text auf der Diskette. Das Gesamtergebnis wird unter dem Namen Neudatei auf dem Laufwerk Null abgespeichert.

Nach der Eingabe des Befehls erscheint ein Fenster mit den angegebenen Ausmaßen. Hier können Sie nun die Eingaben machen, die zu Beginn der Textdatei noch zusätzlich erscheinen sollen. Durch Ctrl und \ wird die Eingabe abgeschlossen und das Gesamtergebnis auf der Diskette unter dem angegebenen Namen abgelegt.

Auf ähnliche Art und Weise kann natürlich auch an das Ende einer bestehenden Textdatei eine Eingabe angehängt werden.

Hierzu müssen hinter dem Join-Befehl nur die ersten beiden Parameter vertauscht werden:

```
join df0:textdatei con:10/10/400/100/Eingabe as df0:neudatei
```

Ein CLI-Wecker

Gerade bei der spannenden Arbeit im CLI kann es schon mal passieren, daß man einen wichtigen Termin einfach vergißt. Natürlich gibt es auf der Workbench-Diskette die Uhr, mit der man auch eine Alarmzeit programmieren kann. Sie hat aber auch einige Nachteile:

Erstens muß sie meistens lange gesucht werden, zweitens braucht sie für ihre Aufgabe zuviel Speicherplatz, drittens können nur absolute Alarmzeiten programmiert werden, und viertens kann man immer nur eine Alarmzeit vorwählen.

Mit einem kleinen Trick kann man sich im CLI diese Uhr sparen. Die folgenden Zeilen demonstrieren, wie das funktioniert:

```
run wait 10 min + (Return)
echo "Hey, der Kaffe ist fertig !" (Return)
```

Leider erscheint die Ausgabe immer in dem CLI, von dem aus der Prozeß gestartet wurde. Man sollte daher darauf achten, daß dieses CLI nie von anderen Fenstern verdeckt wird.

Mit einem weiteren Trick kann aber zumindest erreicht werden, daß alle Screens einmal kurz aufblitzen. Man kann dann in aller Ruhe alle vorhandenen CLI-Fenster nach dem Text absuchen, den die Alarmuhr ausgegeben hat. Um dies zu erreichen, muß der im "Alarmfall" auszugebende Text innerhalb der Anführungszeichen einfach durch die Tastenkombination Ctrl und g erweitert werden. Bei der Eingabe sieht man zwar außer einem kurzen Blitzen des Bildschirms nichts, das Steuerzeichen ist aber tatsächlich im Text vorhanden.

Mit dem Wait-Befehl kann für den Alarm auch eine Uhrzeit vorgewählt werden. Im Kapitel 2 finden Sie zu diesem Befehl alle erlaubten Parameter.

Da die CLI-Uhr einen eigenen Prozeß belegt, können Sie gleich mehrere Wecker auf einmal stellen. Selbst wenn man hiermit die im Amiga maximal mögliche Zahl von 20 Prozessen ausreizt, bemerkt man im CLI dadurch kaum eine Beeinträchtigung in Form einer verlangsamten Rechnergeschwindigkeit.

Umrechnung Tastatur -> ASCII

Besonders beim Programmieren passiert es manchmal, daß man bestimmte ASCII-Codes wissen muß, mit denen ja alle Tastaturzeichen im Amiga intern dargestellt werden. Bevor Sie in Zukunft hierfür Bücher zu Rate ziehen, geben Sie doch einfach vom CLI aus die folgende Zeile ein:

```
type con:300/10/150/50/Converter to * opt h
```

Es erscheint oben rechts auf dem Bildschirm ein kleines Fenster, in dem Sie jetzt die Zeichen eingeben können, deren ASCII-Code Sie benötigen. Die Eingabe muß immer auf 16 Zeichen erweitert werden. Um zum Beispiel den ASCII-Code der Buchstaben S und J zu bestimmen, geben Sie diese beiden Zeichen ein, und betätigen Sie nun so oft die Return-Taste (14mal), bis das Ergebnis im CLI-Fenster erscheint. Für die beiden Buchstaben erhält man so die ASCII-Werte $73 bzw. $6A. Die restlichen Stellen werden mit dem Zeichen für die Return-Taste aufgefüllt ($0A). Die Angaben werden also immer im hexadezimalen Format gemacht (daher auch das opt h!). Beendet wird diese kleine Umrechnungsfunktion durch Ctrl und \.

5.7 Anwendungen zur seriellen Schnittstelle

Die Multitasking-Fähigkeiten des Amiga sind wohl jedem bekannt. Häufig wird in Verbindung mit diesem Rechner aber auch das Wort Multiuser-Betrieb in den Mund genommen. Darunter versteht man das zusammenschalten mehrerer Terminals an eine Zentraleinheit. Ein solches Terminal kann eine beliebige Bildschirmstation sein, die lediglich eingegebene Tastaturzeichen an die Zentraleinheit schickt und auf dem Bildschirm die von der Zentraleinheit empfangenen und ausgewerteten Informatio-

nen wieder ausgibt. Die Aufgabe der Zentraleinheit würde in unserem Fall dem Amiga zukommen, da er ja mehrere Programme getrennt abarbeiten kann.

Die Funktion eines Terminals könnte theoretisch jeder kleine Rechner übernehmen, der mit einer entsprechenden Schnittstelle ausgestattet ist, und so mit der Zentraleinheit kommunizieren kann (z.B. VC20, C64 etc.). Da der Amiga aber nur eine serielle Schnittstelle adressieren kann, ist ohne eine umfangreiche Hardware (oder auch Software) nur der Anschluß eines weiteren Terminals möglich.

Wir haben dies einmal ausprobiert. Die Aufgabe des Terminals hat ein Amiga 500 in der Grundkonfiguration übernommen, die Aufgabe des Zentralrechners ein Amiga 2000 mit Festplatte und zwei Diskettenstationen. Das passende Kabel haben wir selbst gelötet. Es besteht aus zwei 25-poligen Sub-D-Buchsen, die über ein 7-adriges Kabel miteinander verbunden sind. Die Länge des Kabels ist relativ unkritisch, da eine serielle Datenübertragung nicht sehr störanfällig ist. Die Anschlußbelegung sieht folgendermaßen aus:

Pin	mit Pin	Funktion	
2	3	TXD->RXD	(Sender->Empfänger)
3	2	RXD<-TXD	(Empfänger<-Sender)
4	5	RTS->CTS	(Handshake eine Richtung)
5	4	CTS<-RTS	(Handshake andere "")
6	20	DTR->DSR	(Funktionskontrolle)
20	6	DSR<-DTR	(Funktionskontrolle)
7	7	GND	(Masse)

Man erkennt die vollständig symmetrische Pinbelegung. Daher ist es auch egal, welche Buchse man mit welchem Rechner verbindet. Beim Kauf der 25-Buchsen sollte man auf das Gehäuse achten: Die meist grauen Sub-D-Stecker passen aufgrund ihrer Gehäuseschrauben nur schlecht in den Amiga. Besser (und natürlich auch ein wenig teurer) sind die Stecker, die wie das Monitorkabel mit dem Amiga verschraubt werden können. Dieses kann auch an einen Atari-Rechner (der selbstverständlich nur als Terminal dienen kann) angeschlossen werden.

Unser Kabel unterstützt sowohl 3-Draht (xON, xOFF) als auch den RTS/CTS-Handshake.

Achtung: Schalten Sie in jedem Fall vor der Herstellung der Verbindung beide Rechner vollständig ab. Die 8520-Bausteine (Stückpreis immerhin ca. 30 DM) und die vorgeschalteten RS232-Treiber belohnen diese Rücksichtnahme durch ihr weiteres Funktionieren.

Für unsere ersten Versuche im CLI haben wir mit Hilfe des Programms Preferences auf beiden Rechnern die folgenden Parameter eingestellt und abgespeichert:

```
Baud Rate 9600
Buffer Size 512
Read Bits 8
Write Bits 8
Stop Bits 2
Parity None
Handshaking RTS/CTS
```

Der erste vorsichtige Test mit dem Terminalprogramm, das sich auf der dem Amiga mitgelieferten Extras-Diskette befindet und auf beiden Rechnern gestartet wurde, ist leider fehlgeschlagen. Die Meldung "Modem nicht bereit", die dieses Programm auf beiden Rechnern in der obersten Zeile anzeigt, führen wir auf ein Fehlen eines Signals an Pin 8 zurück. Die Bezeichnung für dieses Signal lautet Carrier Detect. Es ist im Modembetreib immer dann aktiv, wenn ein Trägersignal (Piepton) von der Gegenstation empfangen wird: Pech gehabt!

Der nächste Versuch war schon ein wenig erfolgreicher: Auf dem Zentralrechner (Amiga 2000) haben wir vom CLI aus den Befehl eingegeben:

```
run copy ser: to con:10/10/300/100/Empfang
```

Alle empfangenen Daten sollen also in einem Fenster der angegebenen Größe ausgegeben werden. Mit dem 500er haben wir dann das aktuelle Inhaltsverzeichnis auf die serielle Schnittstelle umgeleitet:

```
dir >ser:
```

Ergebnis: Es funktionierte nicht auf Anhieb. Erst nach mehrmaliger Ausgabe des Diskettenverzeichnisses kamen endlich die ersten Daten auf dem 2000er an. Ein Rest blieb aber immer solange im Empfangspuffer verschwunden, bis wir neue Daten hinterhergeschoben haben. Dies ist ein eindeutiger Programmfehler der Schnittstelle, die keine Datei-Endekennzeichen auswertet.

Gut, dieses Manko kann so schnell nicht beseitigt werden. Die Übertragung hat aber im Prinzip geklappt, und daher haben wir uns Gedanken über den Multiuser-Betrieb gemacht. Dabei ist uns für den Zentralrechner (Amiga 2000) eine einfache Lösung eingefallen: Der NewCLI-Befehl kann ja bei Bedarf um eine vollständige Con-Fensterangabe erweitert werden. Einerseits werden aus diesem Fenster dann die Befehle entgegengenommen, und andererseits schreiben die CLI-Befehle ihre Ausgaben in dieses Fenster hinein. Der nun folgende Trick ist genauso einfach wie wirkungsvoll: Der NewCLI-Befehl wird vollständig auf die serielle Schnittstelle umgeleitet. Das geht folgendermaßen:

```
run newcli ser:
```

Mehr nicht! Wenn jetzt die Schnittstelle so arbeiten würde, wie man es normalerweise von ihr erwartet, kann der 2000er über RS232 ein Terminal bedienen.

Leider ist es vom CLI aus nicht so einfach möglich, ein echtes Terminal zu simulieren. Wir haben daher einfach eine kleine Batch-Datei vom 500er an den 2000er geschickt. Da sie lang genug war, sind auch die ersten Befehle von dem CLI-Hintergrundprozeß des 2000ers bearbeitet worden: Plötzlich lief die Festplatte an, da in der Batch-Datei stand: *dir jh0:*. Außerdem zeigte der Status-Befehl die Bearbeitung dieses Befehls an. Der erste Schritt zum Multiuser-Betrieb ist auf jeden Fall gemacht. Erfahrene C-Programmierer werden keine Schwierigkeiten haben, mit der widerborstigen Schnittstelle fertigzuwerden und ein kleines Terminalprogramm zu schreiben. Wer schon im Besitz der neuen Workbench-Diskette (Version 1.3) ist, der kann sich mittels *mount aux:* eine weitere serielle Schnittstelle verschaffen, mit der es die beschriebenen Schwierigkeiten nicht gibt.

Nun haben wir noch eine weitere Anwendung für das Kabel auf Lager. Ändern Sie doch einmal den Aufruf, mit dem Sie die Workbench laden, folgendermaßen um:

```
loadwb -debug
```

Nach dem Start der Workbench sieht noch alles völlig normal aus. Halten Sie aber einmal die rechte Maustaste gedrückt und bewegen sie über den rechten Rand der Menüleiste hinaus. Irgendwann taucht plötzlich ein völlig neues Menü auf, das aus zwei Unterpunkten besteht:

1. Debug
2. Flushlibs

Es ist nun möglich, von einem zweiten Rechner aus den sogenannten ROM-Wack (ein Diagnoseprogramm) aufzurufen, falls ein Task Held oder eine Guru-Meditation auftritt.

Als einfaches Terminalprogramm genügt schon das BASIC-Programm Kommunik, das sich auf der Extras-Diskette in dem Ordner Basic Demos befindet. Sollte sich der Rechner aufhängen und man betätigt die rechte Maustaste (Press right button to cancel/debug), kann von einem angeschlossenen Rechner mit dem Terminalprogramm der Speicher des abgestürzten Rechners inspiziert werden. Der Menüpunkt Debug schaltet den ROM-Wack auch dann ein, wenn kein Absturz stattgefunden hat. Durch den Menüpunkt Flushlibs wird soviel Systemspeicher wie möglich als belegt gekennzeichnet, damit vorhandene Strukturen (Librarys, Devices etc.) nicht überschrieben werden können.

Zur Bedienung des ROM-Wacks nun ein Beispiel: Rechner A und Rechner B sind über das RS232-Kabel verbunden. Durch die folgende Befehlsfolge wird nun in Rechner B ein Task-Held-Zustand erzeugt:

```
mount res0:
dir >res0:
```

Das Devive res0: befindet sich meist zu Demonstrationszwecken in der Datei MountList des Devs-Verzeichnisses. Bindet man

dieses Device aber tatsächlich in den Amiga ein und leitet Ausgaben darauf, tritt normalerweise der Task-Held-Zustand ein.

Betätigt man nun die rechte Maustaste, gibt es eine echte Guru-Meditation. Ein weiterer Druck auf diese Taste bewirkt auf dem Terminalrechner die Ausgabe:

```
rom-wack
PC: FE66E8 SR: 0015 USP: 022B32 SSP: 07FFF2 XCPT: 0003 TASK: 0221A0
DR: 0000FFFF 0000FFFF 0001C815 00000001 00000000 00022394 000221FC
00000019
AR: 000231B5 ABABABAB 00023090 00022394 00FF4834 00005D4A 000045B8
SF: 2271 ABAB ABDD 2269 0015 00FE 66E8 0000 0000 0000 0676 00FC 07DC
00FC 07DE
```

Assembler-Experten können diese Angaben nun auswerten und über die vom Wack bereitgestellten Funktionen an dem abgestürzten Rechner noch Manipulationen vornehmen. Ein Druck auf das Fragezeichen des Terminalrechners läßt eine Liste aller Befehle erscheinen, die ROM-Wack kennt:

```
alter  boot  clear  fill  find  go  ig  limit  list  regs
reset resume set show user
```

Hier die wichtigsten Kommandos:

Alter

Dient zum Ändern von Speicherinhalten.

Boot

Der abgestürzte Rechner wird neu gebootet.

Clear

Speicherbereiche löschen.

Fill

Speicherbereiche füllen.

Find

Speicherbereiche nach bestimmten Hex-Werten absuchen.

Go

Programm starten.

Regs

Ausgabe der Registerinhalte.

Reset

Verhält sich wie "boot".

Resume

Falls Debug von der Workbench angewählt wurde, wird die Workbench hierduch wieder aktiviert.

Der Adreßbereich wird verändert, indem man einfach eine neue Adresse (ohne jeden Befehl) eingibt.

Wir haben uns einmal an der Task-Struktur entlanggehangelt, die in dem Beipiel bei $0221A0 beginnt (siehe Registeranzeige unter TASK). Der Befehl lautete also einfach:

```
"0221a0"
```

Dies ergab die folgende Meldung:

```
0221A0   0000   0810   0000   080C   0D0A   0002   2BB6   0002
....^H^P....^H^L^M^J..^B +....^B
```

Mit Hilfe des "Amiga Intern" sind wir dann auf die Adresse gestoßen, die den Zeiger auf den Namen der abgestürzten Task enthält. Es ist die Adresse 022BB6 aus der obigen Zeile. Hier steht nun:

```
022BB6  5245  5330  0000  0000  0000  0002  3338  0000    R E S
0.............. 3 8....
```

Es war also tatsächlich die Task RES0, die dem Amiga zum Verhängnis geworden ist. Interessant wird der ROM-Wack natürlich erst, wenn ein Programm aus unbekannten Gründen dauernd direkt in eine Guru-Meditation springt. Im Task-Held-Zustand könnte der Speicher des Amiga noch jederzeit von einem Monitorprogramm untersucht werden, das vorher von einem zweiten CLI-Prozeß aus gestartet wurde.

Die interessante ROM-Wack-Funktion zeigt sehr deutlich, daß die RS232-Schnittstelle auch so funktionieren kann, wie man es von ihr erwartet. Die Datenübertragung mit 9600 Baud funktioniert tadellos.

6. Batchprocessing auf dem Amiga

Die Stapeldateiverarbeitung, im Neudeutschen auch Batchprocessing genannt, ist auf allen derzeit bekannten DOS-Rechnern zu finden. Auch das AmigaDOS ist hiervon nicht ausgenommen. Dieses Kapitel informiert Sie darüber, was es mit dieser Technik auf sich hat und wie man sie auf dem Amiga realisiert.

6.1 Einführung in die Stapeldateiverarbeitung

Die nun folgenden Abschnitte sollen Sie zunächst einmal mit der Stapeldateiverarbeitung auf dem Amiga vertraut machen. Was man letztendlich damit anfangen kann, werden Sie im Anschluß an diese Einführung sehen.

6.1.1 Was sind Batch-Files?

Im Grunde genommen stellen die CLI-/Shell-Befehle schon eine eigene kleine Programmiersprache dar. Man kann zum Beispiel den Echo-Befehl vergleichen mit der Print-Anweisung der Sprache BASIC. Das Problem ist nur, daß die CLI-/Shell-Befehle nur im Direktmodus eingegeben werden können. Wie im Kommandofenster des AmigaBASIC wird der Befehl nicht als neue Programmzeile in einer Liste abgelegt, sondern nach Return sofort ausgeführt. Die wichtigste Eigenschaft einer Programmiersprache fehlt also noch: Die Befehle können nicht gespeichert und erst auf ein bestimmtes Kommando hin nacheinander ausgeführt werden. Die Aufgabe der Befehlsspeicherung übernehmen im Amiga die sogenannten Batch-Files (Stapeldateien). Bei ihnen handelt es sich um reine Textdateien, die von einem beliebigen Textbe- oder -verarbeitungsprogramm erstellt werden können und die nur aus einer Abfolge von CLI-/Shell-Befehlen bestehen. Ob Sie solche Batch-Dateien mit dem Notepad, Textomat oder den CLI-/Shell-Editoren erstellen, bleibt ganz Ihnen überlassen. Wichtig ist hierbei nur, daß der

Text aus den üblichen (ASCII-) Zeichen besteht. Die Datei kann dann unter einem beliebigen Namen abgespeichert werden.

6.1.2 Wie sehen Batch-Files aus?

Die einfachsten Batch-Dateien bestehen aus den ganz gewöhnlichen CLI-Befehlen, wie man sie auch im Dialogbetrieb (also von CLI oder Shell aus) verwenden kann. In jeder Zeile darf nur ein solcher Befehl stehen. Kommentare sind jedoch zugelassen. Sie müssen durch ein Semikolon eingeleitet werden und sind in ihrer Länge auf die jeweilige Zeile beschränkt.

Eine sehr einfache Datei könnte z.B. so aussehen:

```
copy test#? testprogramm   ; kopiert alle Testversionen
delete test#?              ; in einen speziellen Ordner und löscht
                           ; sie aus dem Hauptverzeichnis.
```

Außer den CLI-/Shell-Befehlen dürfen in Batch-Dateien aber auch noch eine Reihe spezieller Kommandos auftauchen, deren Verwendung in CLI-Fenster oder Shell kaum einen Sinn ergibt oder einfach verboten ist. Es sind die Befehle:

```
echo
failat
quit
if/else/endif
skip/lab
ask
wait
```

Gäbe es diese Befehle If, Else und EndIf nicht, würde eine Batch-Datei immer linear von oben nach unten abgearbeitet. Diese zusätzlichen Befehle ermöglichen es, auf das Auftreten bestimmter Situationen mit einem veränderten Programmablauf zu reagieren. Eine ausführliche Behandlung dieser wie auch der anderen aufgeführten Befehle finden Sie im Kapitel 2.

6.1.3 Wie werden Batch-Files aufgerufen?

Für die Ausführung von Batch-Dateien gibt es den Befehl Execute. Auch er wird ausführlich im Kapitel 2 besprochen. Im einfachsten Fall wird hinter den Execute-Befehl nur der Name der Batch-Datei gehängt, die ausgeführt werden soll. Beispiel:

```
execute df0:meinbatch
```

Ab der Version 1.3 der Workbench besteht die Möglichkeit, das Status-Bit s einer Datei zu setzen. Dies geschieht mit dem Befehl: protect df0:meinbatch +s. Ist das s-Bit gesetzt, dann muß der Execute-Befehl nicht mit eingegeben werden, er muß aber im c-Verzeichnis der Workbench unter seinem vollen Namen vorhanden sein. Der Befehlsname Execute darf also nicht in zum Beispiel Ex umbenannt worden sein. Es reicht dann aus, den Pfad und den Namen der Batch-Datei einzugeben. In unserem Beispiel würde dann die Eingabe von df0:meinbatch genügen, um die Datei abzuarbeiten. Anschließend werden alle CLI-/Shell-Befehle, die sich innerhalb der Datei mit dem Namen meinbatch befinden, der Reihe nach abgearbeitet. Erst danach ist das CLI bzw. die Shell wieder bereit, neue Befehle auszuführen. Sehr praktisch ist häufig auch die Verwendung eines eigenen Hintergrundprozesses für die Abarbeitung von Batch-Dateien. Das obige Kommando muß hierzu nur durch den Befehl Run aufgerufen werden:

```
run execute df0:meinbatch
```

Das Betriebssystem vergibt anschließend für den Execute-Befehl einen eigenen Prozeß, so daß während der Abarbeitung der Batch-Datei in CLI oder Shell weitergearbeitet werden kann. Lediglich eventuelle Ausgaben erscheinen im Ausgabefenster, da ja einem nichtinteraktiven Hintergrundprozeß kein eigenes Ausgabefenster zur Verfügung gestellt wird.

6.1.4 Ein einfaches Beispiel

Zum Abschluß dieser kleinen Einführung soll anhand eines einfachen Beispiels Schritt für Schritt demonstriert werden, wie

eine Batch-Datei erstellt und eingesetzt wird. In den meisten Fällen wird man zur Erstellung einer Batch-Datei auf den ED-Editor zurückgreifen. Dieses kleine Programm gehört zu den normalen CLI-/Shell-Befehlen und ermöglicht eine relativ komfortable und schnelle Bearbeitung kleinerer Texte. Für Batch-Dateien ist dies genau das richtige Werkzeug. Wer lieber mit einem anderen Editor oder einer Textverarbeitung arbeiten möchte, kann auch dies tun. Die Aufgabe lautet: Schreiben Sie eine Batch-Datei, die die Namen aller typischen Batch-Befehle auf dem Bildschirm ausgibt. Die Datei soll den Namen Kommandos bekommen. Lösung: Zuerst muß natürlich einmal ein CLI oder eine Shell geöffnet werden. Der ED wird anschließend aufgerufen durch: ED Kommandos.

Nach einer Weile erscheint das leere Eingabefenster des ED. Da es eine Datei mit diesem Namen noch nicht gibt, steht unten links in der Ecke die Meldung "Creating new file". Nun kann man die geforderten Befehle eintragen:

```
;Kommandos
echo "execute"
echo "echo"
echo "failat"
echo "quit"
echo "if/then/else"
echo "skip/lap"
echo "ask"
echo "wait"
```

Anschließend wird der ED durch die Befehlssequenz ESC X verlassen und die Datei unter dem angegebenen Namen in dem aktuellen Verzeichnis abgespeichert. Nun kann diese Batch-Datei jederzeit durch die Eingabe des Befehls *Execute Kommandos* ausgeführt werden. Ist mit *protect Kommandos +s* das Script-Bit gesetzt worden, reicht auch die Eingabe von *Kommandos* aus, um die Datei auszuführen. Das Ergebnis im Ausgabefenster sieht dann so aus:

```
execute
echo
failat
quit
if/then/else
```

```
skip/lap
ask
wait
```

Wenn man keine größeren Korrekturen bei der Eingabe eines Textes vornehmen muß, kommt man bei der Bedienung des ED mit der verwendeten ESC-Sequenz aus. Im Abschnitt 2.4.1 wird die Bedienung des ED-Editor ausführlicher besprochen.

6.2 Modifikationen an der Startup-Sequence der Workbench 1.2

Im Laufe dieses Kapitels haben Sie erfahren, was eine Batch-Datei ist und was sie bewirkt. Für viele, die den Begriff Batch-Datei zum ersten Mal gehört haben, ist es verwunderlich, wenn wir nun behaupten, daß auch Sie schon mit einer solchen Batch-Datei gearbeitet haben. Ja, wir gehen sogar soweit und behaupten, daß alle diejenigen, die schon einmal, nachdem der Amiga eingeschaltet wurde, die Workbench-Diskette ins Laufwerk df0: gelegt haben, mit einer Batch-Datei gearbeitet haben. Sie wissen es nur vielleicht nicht.

Um Ihnen das zu erklären, bitten wir Sie, eine Kopie Ihrer original Workbench-Diskette anzufertigen. Wenn im weiteren Verlauf dieses Kapitels von der Workbench-Diskette geschrieben wird, dann ist ab sofort immer die Kopie gemeint. Die Originaldiskette sollten Sie immer an einem sicheren Ort aufbewahren.

Doch nun zur Erklärung: Schauen Sie sich dazu bitte einmal das Inhaltsverzeichnis des Unterdirectorys S der Workbench-Diskette an. Öffnen Sie dazu ein CLI-/Shell-Fenster, und legen Sie die Workbench-Diskette in das Laufwerk DF0:.

Geben Sie nun den Befehl Dir df0:S ein. Sie sehen, daß dort eine Datei Namens Startup-Sequence vorhanden ist. Bevor Sie sich nun diese Datei anschauen, ist es sinnvoll, das CLI-/Shell-Fenster auf die maximale Größe aufzuziehen. Geben Sie nun den Befehl *Type df0:S/Startup-Sequence* ein. Nachdem das Laufwerk kurzzeitig angelaufen ist, wird der Inhalt der Datei auf dem Bildschirm sichtbar. Damit der Text nicht in einem

Zuge durchläuft, empfehlen wir Ihnen, die Maus direkt nach dem Drücken der Return-Taste griffbereit zu haben. Indem Sie die rechte Maustaste gedrückt halten, ist es möglich, das Weiter-Scrollen des Textes solange zu unterbinden, wie Sie es für nötig halten. Sie müßten nun auf Ihrem Bildschirm, je nachdem, welche Ausführung des Amiga Sie haben, einen Text vor sich sehen, der in etwa so aussieht (Es handelt sich hierbei um die Startup-Sequence des Amiga 2000. Die Sequence des Amiga 500 ist wesentlich kürzer gehalten.):

```
BindDrivers
echo "A2000  Workbench 1.2D  V33.56  27-APR-87*N"
Assign WB: sys:
IF EXISTS sys:s/JHO
   echo "Initializing Janus ...*N"
   Wait 25
   Echo "Mounting Janus Hard Disk*N"
   DJMount
   IF EXISTS JHO:
      Assign WB: JHO:
      Echo "Transfering control to JHO: *N"
   ELSE
      Assign WB: sys:
   EndIF
EndIF
IF EXISTS sys:s/DHO
   Assign WB: DHO:
   Echo "Transfering control to DHO: *N"
EndIF
AddBuffers WB: 30
CD WB:
IF EXISTS WB:c
   Assign c: WB:c
EndIF
IF EXISTS WB:s
   Assign s: WB:s
EndIF
IF EXISTS WB:t
   Assign t: WB:t
EndIF
IF EXISTS WB:l
   Assign l: WB:l
EndIF
IF EXISTS WB:libs
   Assign libs: WB:libs
EndIF
IF EXISTS WB:devs
   Assign devs: WB:devs
EndIF
IF EXISTS WB:fonts
```

```
   Assign fonts: WB:fonts
EndIF
Assign sys: WB:
IF EXISTS WB:System
   Assign System: WB:System
   Path add System
EndIF
IF EXISTS WB:Utilities
   Assign Utilities: WB:Utilities
   Path add Utilities
EndIf
IF EXISTS WB:PC
   Path add WB:PC
EndIf
Assign WB:
Dir RAM:
Path RAM: add
failat 30
SetClock opt load
SetMap d
LoadWb
IF Exists sys:s/Startup-HD
   Execute sys:s/Startup-HD
EndIF
endcli > nil:
```

Wenn Sie sich diese Datei genau angeschaut haben, dann ist Ihnen vielleicht aufgefallen, daß in diesem Text nur CLI-/Shell-Befehle benutzt wurden. Es handelt sich hierbei also um eine Batch-Datei. Einschränkend müssen wir allerdings hinzufügen, daß es sich um eine ganz spezielle Batch-Datei handelt. Denn die Befehle dieser Datei werden, sofern sie vorhanden sind, nach jedem Neustart des Amiga automatisch ausgeführt. Das heißt, daß Sie schon mit einer Batch-Datei gearbeitet haben, ohne daß Sie es wollten. Gehen wir doch nun etwas genauer auf diese Startup-Sequence ein. Durch diese Datei haben Sie die Möglichkeit, dem Rechner mitzuteilen, in welcher Form er sich beim "Hochfahren" zu melden hat und welche Einstellungen er vornehmen soll. Wenn nun im folgenden die Startup-Sequence von Ihnen abgeändert wird, beachten Sie, daß Sie Änderungen nur auf der Kopie der Workbench vornehmen. Falls Sie durch eventuelle Tippfehler eine für den Rechner unbrauchbare Startup-Sequence auf Ihrer Diskette haben, dann ist es sinnvoll, daß Sie noch eine brauchbare Workbench in Reserve haben. Doch jetzt gehen wir zum Abändern über.

Wie Sie sehen, ist im ersten Teil der Sequence von dh0 und jh0 die Rede. Die dort vorgenommenen Einstellungen sind nur für die Amiga-Besitzer interessant, die entweder eine Festplatte oder eine XT-/AT-Karte in Verbindung mit einer Festplatte haben. Da der größte Teil der Amiga-Besitzer, insbesondere die Amiga-500-Besitzer, jedoch weder eine Festplatte noch eine XT-/AT-Karte haben, möchten wir Ihnen zeigen, wie Sie die Startup-Sequence für einen "einfachen" Amiga ohne Festplatte oder XT-/AT-Karte kürzer gestalten können. Nun könnten Sie fragen, was es für einen Sinn hat, diese Datei zu kürzen. Je weniger Befehle abgearbeitet werden müssen, umso schneller der Amiga für Sie arbeitsbereit. Deshalb übernehmen wir aus dem angesprochenen ersten Teil der Sequence nur den Befehl Assign WB: SYS:. Auf die Ausgabe des Textes in der zweiten Zeile können wir im Hinblick auf die Zeitersparnis getrost verzichten. Auch die beiden nachfolgenden großen If-EndIf-Bereiche werden, da sie nur für Festplattenbesitzer interessant sind, samt Inhalt weggelassen. Die nächsten für uns wichtigen Befehle sind somit: *AddBuffers WB: 30* und *CD WB:*. Was die Befehle bewirken, können Sie im Kapitel 2 nachschlagen.

Es folgen nun eine Reihe von Assign-Befehlen, die jeweils von einem If-/EndIf-Block umrahmt werden. Auch hier können wir einige Befehle einsparen. Da es sich bei den If-Fragen jeweils um Sicherheitsabfragen handelt, können wir dem Rechner hier einiges an Arbeit abnehmen. Dazu müssen Sie feststellen, ob sich folgende Directorys auf Ihrer Workbench-Diskette befinden: C, S, T, L, Libs, Devs, Fonts, System und Utilities. Um dies nachzuprüfen, geben Sie einfach den Befehl Dir df0: ins CLI bzw. die Shell ein. Unter den anschließend auf dem Bildschirm aufgelisteten Files müssen sich nun die oben aufgeführten Directorys befinden. Ob sich zusätzlich zu den Directorys auch noch die entsprechenden .info-Files in dieser Liste befinden, spielt keine Rolle. Wenn Sie also festgestellt haben, daß die erforderlichen Directorys vorhanden sind, können die Sicherheitsabfragen in der Startup-Sequence auch entfernt werden. Das heißt, daß Sie sowohl vor als auch nach den Assign-Befehlen die entsprechenden If- und EndIf-Zeilen löschen dürfen. Es bleiben demnach folgende Zeilen aus dem mittleren Teil der Sequence erhalten:

```
Assign c: WB:c
Assign s: WB:s
Assign t: WB:t
Assign l: WB:l
Assign libs: WB:libs
Assign devs: WB:devs
Assign fonts: WB:fonts
Assign sys: WB:
Assign System: WB:System
Path add System
Assign Utilities: WB:Utilities
Path add Utilities
```

Falls bei Ihnen nun die Frage nach der Zeile Path add WB:PC auftritt, möchten wir Sie darauf aufmerksam machen, daß in der obigen Directory-Liste das Vorhandensein des PC-Directorys nicht notwendig war. Der Grund dafür ist der, daß für die Amiga-Besitzer, die keine XT-/AT-Karte in ihrem Amiga haben, dieses Directory absolut überflüssig ist. Deshalb müssen Sie dieses Directory in der Startup-Sequence auch nicht berücksichtigen. Nebenbei bemerkt, können Sie durch Löschen dieses PC-Directorys und dessen .info-File ein wenig Platz auf Ihrer Workbench schaffen. Möchten Sie noch mehr freien Speicherplatz auf der Workbench-Diskette haben, dann können Sie auch noch die Directorys Sidecar und Expansion samt deren .info-Files bedenkenlos löschen. Falls Sie sich später doch noch eine Festplatte oder eine XT-/AT-Karte zulegen möchten, dann haben Sie immer noch Ihre original Workbench-Diskette, auf der diese Dateien weiterhin enthalten sind.

Doch wieder zurück zur Startup-Sequence. Es folgt der Befehl Assign WB:. Auch er wird in die neue Startup-Sequence übernommen. Bei der nächsten Zeile können Sie selber entscheiden, ob sie gelöscht oder übernommen werden soll.

Möchten Sie die Zeile löschen, dann haben Sie, nachdem die Startup-Sequence abgearbeitet worden ist, kein Disketten-Icon der RAM-Disk auf Ihrer Workbench-Benutzeroberfläche. Behalten Sie die Zeile bei, erscheint sofort das Disketten-Icon der RAM-Disk auf dem Bildschirm.

Wir haben uns gegen das Disketten-Icon entschieden und lassen dementsprechend die obige Zeile aus der neuen Startup-Se-

quence heraus. Der darauffolgende Path RAM: ADD-Befehl ist für uns ebenfalls unwichtig, er wird also nicht übernommen.

Eine Frage zwischendurch: Ist es für Sie wichtig zu wissen, wie spät es ist, wenn Sie Ihren Amiga einschalten? Nein? Dann fällt es Ihnen auch nicht schwer, die Zeile *SetClock opt Load* in der neuen Sequence zu streichen. Wenn doch, dann übernehmen Sie die Zeile unverändert. Auch die folgende Zeile könnte wegfallen. Dies hat allerdings den Nachteil, daß Sie anschließend die amerikanische Tastaturbelegung vorliegen haben, das heißt, daß das Y auf der Z-Taste zu finden ist, das Semikolon auf der Ö-Taste usw. Die Zeile *Setmap D* muß folglich unbedingt in der neuen Sequence enthalten sein.

Anschließend wird mit LoadWB die eigentliche Benutzeroberfläche, die Workbench, geladen. Auch sie sollte übernommen werden. Anschließend folgt wieder eine Sicherheitsabfrage. Es wird gefragt, ob eine Startup-HD-Datei im Directory S vorhanden ist. Erinnern Sie sich noch, welche Dateien aufgelistet wurden, nachdem Sie den Befehl Dir df0:S eingegeben haben? Unter den Dateien befindet sich eine, die uns direkt eine Antwort auf unsere obige Frage gibt. Wenn Sie sich nicht erinnern, dann geben Sie den Befehl einfach noch einmal ein. Die Datei, die ich meine, heißt No-hd. Es gibt also keine Startup-HD-Datei. Folglich benötigen wir in der neuen Sequence weder die Sicherheitsabfrage noch den darin eingeschlossenen Execute-Befehl. Die drei Zeilen werden also nicht übernommen.

Damit das CLI-Fenster nach Abarbeitung der Startup-Sequence geschlossen wird, nehmen wir auch die Zeile *EndCLI > NIL:* in die neue Sequence hinein. Nachdem wir nun besprochen haben, was die neue Startup-Sequence alles enthalten sollte, möchte ich Ihnen diese Sequence komplett mit allen Befehlszeilen zeigen.

```
;neue_Startup-Sequence
Assign WB: sys:
Addbuffers WB: 30
CD WB:
Assign c: WB:c
Assign s: WB:s
Assign t: WB:t
Assign l: WB:l
```

```
Assign libs: WB:libs
Assign devs: WB:devs
Assign fonts: WB:fonts
Assign sys: WB:
Assign Utilities: WB:Utilities
Path add Utilities
Assign System: WB:System
Path add System
Assign WB:
SetMap d
LoadWb
endcli > nil:
```

Alle wichtigen Befehle der alten Sequence sind in dieser neuen ebenfalls enthalten. An diese neue Startup-Sequence ist allerdings eine Forderung geknüpft. Es muß gewährleistet sein, daß die oben angesprochenen Directorys jederzeit auf der Workbench-Diskette enthalten sind. Ist dies nicht gewährleistet, kann es beim Starten zu Schwierigkeiten kommen. Doch was hat es für Auswirkungen, wenn Sie die neue anstelle der alten Startup-Sequence benutzen? Wir haben uns die Mühe gemacht und die Zeiten von beiden Sequencen miteinander verglichen. Das Ergebnis war, daß die neue Sequence um ca. 50% schneller ist. Sie sparen also bei jedem erneuten Hochfahren die Hälfte der ansonsten benötigten Zeit ein. Es ist unserer Meinung nach lohnenswert, sich diese einmalige Arbeit zu machen, um anschließend von der Zeitersparnis profitieren zu können.

Wenn Sie einen Amiga 500 besitzen, dann sehen Sie, daß in der Startup-Sequence die Assign-Befehle nicht vorhanden sind. Übernehmen Sie dementsprechend nur die anderen Befehle in die neue Sequence. Die Assign-Befehle müssen Sie nicht unbedingt ergänzend hinzufügen.

Um die neue Sequence herzustellen, ist es noch nicht einmal notwendig, alles von Grund auf neu einzutippen. Es geht viel schneller, wenn Sie die alte Sequence einfach abändern. Wir haben die Erfahrung gemacht, daß es am einfachsten ist, diese Änderungen mit dem auf der Workbench-Diskette vorhandenen ED-Editor vorzunehmen.

Gehen Sie dazu wie folgt vor: Behalten Sie weiterhin die Kopie der Workbench-Diskette im Laufwerk df0:. Geben Sie dann den

Befehl *ED df0:S/Startup-Sequence* in CLI/Shell ein. Daraufhin wird das Laufwerk kurzzeitig anlaufen. Der Editor wird gestartet, und die alte Startup-Sequence wird direkt in den Editor geladen. jetzt müssen Sie nur noch mit Hilfe der Cursor-Tasten den Cursor so positionieren, daß er in einer Zeile steht, die nicht in die neue Sequence übernommen werden soll. Dabei ist es egal, wo in der Zeile der Cursor steht. Um diese Zeile zu löschen, genügt es, die Ctrl-Taste und die B-Taste gleichzeitig zu drücken. Verfahren Sie mit den anderen zu löschenden Zeilen ebenso. Haben Sie alle Zeilen gelöscht, die nicht in die neue Sequence gehören, müssen Sie den so entstandenen Text wieder abspeichern. Dazu betätigen Sie die ESC-Taste, und anschließend geben Sie ein X ein. Nach dem abschließenden Return läuft das Laufwerk erneut an, und die neue Startup-Sequence wird anstelle der alten Sequence auf der Diskette abgespeichert. Wenn Sie es ausprobieren, werden Sie feststellen, daß es gar nicht so kompliziert ist, wie es sich hier vielleicht anhört.

Aber Sie können es sich noch einfacher machen, indem Sie die Datei neue_Startup-Sequence, die sich auf der Diskette, die in diesem Buch ist, befindet, als Ihre Startup-Sequence in das s-Verzeichnis Ihrer Workbench-Diskette kopieren. Vorher sollten Sie die originale Startup-Sequence umbenennen, damit diese nicht gelöscht wird und Sie sie im Bedarfsfall zu Ihrer Verfügung haben. Die Befehle dazu lauten:

```
rename sys:s/Startup-Sequence as sys:s/Startup-Sequence_original
copy AmigaDOS-Buch:Batches/neue_Startup-Sequence
sys:s/Startup-Sequence
```

Nun haben Sie eine schnellere Startup-Sequence. Wie Sie die Startup-Sequence noch ein wenig modifizieren können, lesen Sie in dem folgenden Kapitel.

6.2.1 Die komfortable Startup-Sequence der Workbench 1.2

Sie haben im vorigen Kapitel erfahren, wie Sie es ermöglichen können, Ihre Startup-Prozedur zu beschleunigen. Wir möchten im folgenden ein wenig vom Aspekt der Zeitersparnis übergehen zu mehr Anwenderfreundlichkeit. Das heißt, daß Ihre Startup-

Sequence wieder einige Befehlszeilen länger wird. Am Ende müssen allerdings Sie entscheiden, auf was Sie mehr Wert legen, auf Schnelligkeit oder Benutzerfreundlichkeit.

Da wir wieder eine Startup-Sequence abändern möchten, brauchen wir erst einmal eine Grundlage. Wir haben uns die verkürzte Version aus dem Kapitel 6.2 ausgesucht und legen diese somit zugrunde. Wir möchten Sie aber darauf aufmerksam machen, daß auch die Startup-Sequence des Amiga 500 in gleicher Weise abgeändert werden kann.

Bevor wir jetzt die Sequence ändern, noch eine Frage. Kennen Sie den Ask-Befehl? Falls Sie diese Frage mit *Nein* beantworten, dann schauen Sie besser noch einmal in das Kapitel 2 hinein.

So, wir hoffen, daß nun jeder von Ihnen weiß, was man mit diesem Befehl bewirken kann und wie man ihn einsetzt. Doch noch ein Hinweis. Wir möchten Sie bitten, auf Ihrer Workbench-Diskette im Verzeichnis C nachzuschauen, ob Sie den Befehl überhaupt haben. Wir mußten feststellen, daß verschiedene Workbench-Versionen diesen Befehl nicht enthalten. Ist dies bei Ihnen der Fall, dann bitten wir Sie, bei Ihrem Händler oder bei Ihren Bekannten nachzufragen, ob sie diesen Befehl auf ihrer Workbench-Diskette haben. Haben Sie den Befehl auf einer anderen Workbench gefunden, dann müssen Sie ihn nur noch in Ihr C-Verzeichnis kopieren. Im weiteren Verlauf dieses Buches gehen wir davon aus, daß Sie den Ask-Befehl zu Ihrer Verfügung haben.

Doch kommen wir nun zur Startup-Sequence zurück. Haben Sie sich auch schon einmal darüber geärgert, daß Sie nach dem Starten des Rechners zuerst die verschiedensten Icons anklicken mußten, bevor Sie ein benutzbares CLI-/Shell-Fenster zu Verfügung hatten? Sie können dieses Problem umgehen, indem Sie die Zeile *EndCLI > NIL:* einfach weglassen. Eleganter lösen Sie allerdings dieses Problem, indem Sie den Ask-Befehl in Ihrer Startup-Sequence benutzen. Wir möchten Ihnen einen Vorschlag machen, wie so etwas aussehen könnte. Fügen Sie dazu die folgenden Zeilen vor der letzten Zeile in Ihre Startup-Sequence ein.

```
ask "Möchten Sie ein CLI-Fenster offen halten? (y/n) "
if warn
   ask "Möchten Sie ein großes (y) oder ein kleines (n)
CLI-Fenster? "
       if warn
          newcli "con:0/0/550/200/Startup-CLI"
       else
          newcli "con:0/0/160/30/Startup-CLI"
       endif
endif
```

Haben Sie dies getan, dann können Sie sich die Auswirkungen dieser Zeilen anschauen, indem Sie die beiden Amiga-Tasten und die Ctrl-Taste gleichzeitig drücken, also einen sogenannten Warmstart durchführen.

Es wird wieder die Startup-Sequence abgearbeitet. Bevor sie allerdings komplett abgearbeitet wurde, wird Ihnen eine Frage gestellt: "Möchten Sie ein CLI-Fenster offen halten? (y/n)."

Die beiden Buchstaben Y und N sollen Ihnen vorgeben, welche beiden Antwortmöglichkeiten Sie haben. Es funktioniert aber auch mit yes und no. Beantworten Sie die Frage mit no, dann geschieht nichts weiter. Das CLI-Fenster wird geschlossen, und Sie sind gezwungen, mit der Maus weiterzuarbeiten. Tippen Sie hingegen yes ein, dann werden Sie wieder gefragt. Diesmal allerdings wird gefragt, ob Sie ein großes oder ein kleines CLI-Fenster haben möchten. Unabhängig davon, ob Sie Yes oder No eingeben, wird ein weiteres CLI-Fenster mit dem Namen Startup-CLI geöffnet. Haben Sie yes eingegeben, dann fällt dieses CLI recht groß aus. Sie haben dann die Möglichkeit, sofort im CLI zu arbeiten. Bei No hält sich die Größe dagegen in Grenzen. Man kann sogar behaupten, daß es recht winzig ist. Jedenfalls viel zu klein, als daß man sofort mit diesem Fenster arbeiten kann. Dies ist aber beabsichtigt. Dieses Fenster soll lediglich als Reserve auf dem Bildschirm vorhanden sein. Im eventuellen Bedarfsfall müssen Sie das Fenster nur noch aufziehen und können sofort mit der Arbeit im CLI beginnen. Ob Sie diesen Zusatz in Ihre Sequence aufnehmen, bleibt Ihnen überlassen. Wir wollten Ihnen damit nur zeigen, daß auch so etwas möglich ist. Dies soll Ihnen nur als Anregung dienen. Sie können dieses Prinzip bei anderen Dingen ebenso einsetzen. Zum Beispiel könnten Sie auch abfragen, ob die Arbeitsoberfläche, die Work-

bench, geladen werden soll oder nicht. Wenn Sie ausschließlich mit dem CLI arbeiten, brauchen Sie die Icons nicht. Auch hier läßt sich dies einfach mittels des Ask-Befehls realisieren.

```
ask "Soll die Workbench geladen werden? (y/n) "
if warn
   loadwb
endif
```

Die in der Startup-Sequence enthaltene LoadWB-Zeile müßten Sie nur gegen die obigen vier Zeilen ersetzen. Die beiden obigen Beispiele, die Startup-Sequence abzuändern, stellen nur einen kleinen Teil dessen dar, was möglich ist. Sie sind mit dem oben dargelegtem Prinzip in der Lage, die Sequence nach Ihren Wünschen und Erfordernissen zu gestalten.

Wir möchten Sie im Hinblick auf weitere Ideen dieser Art auf das Kapitel 6.3 aufmerksam machen. Dort finden Sie eine Batch-Datei, die gut in die Startup-Sequence eingebunden werden könnte.

6.2.2 Die Startup-Sequencen der Workbench 1.3

Wie aus der Überschrift ersichtlich, handelt es sich nicht nur um eine Sequence, sondern um mehrere. Wir besprechen als erstes die vom Namen her bereits bekannte Startup-Sequence. Sie hat die gleiche Funktion wie bei der Version 1.2, sieht aber etwas anders aus:

```
c:run >NIL: c:SetAlert ;patch system
cd c:
echo "A500/A2000 Workbench disk.   1.3 version *N"
Sys:System/FastMemFirst ; move C00000 memory to last in list
BindDrivers
Addbuffers df0: 10
FF >NIL: -0 ; start FastFonts
MakeDir Ram:T ; set up logical T: directory for Execute
Assign T: ram:T
MakeDir Ram:Env ; set up ENV: directory
Assign ENV: ram:Env
path sys:utilities sys:system ram: sys: s: add
Sys:System/SetMap usa1
LoadWb
SetClock >NIL: load ;load system time from real time clock
```

```
resident CLI L:Shell-Seg SYSTEM pure ; activate Shell
resident c:Execute pure ; pre-load the Execute command
mount newcon: ; mount the improved console handler
endcli > nil:
```

Auch hier könnte die ein oder andere Zeile entfernt werden. Dazu besprechen wir die einzelnen Zeilen der Reihe nach. Fangen wir also bei der ersten an. Sie sollte auf jeden Fall nicht gelöscht werden, da hier das System "gepatched" wird, das heißt das System wird ein klein wenig modifiziert. Zu dem eigentlichen Befehl SetAlert sei noch erwähnt, daß er nur einmal eingegeben werden sollte. Und da dies schon in der Startup-Sequence der Fall ist, sollte dieser Befehl niemals noch einmal in CLI oder Shell eingegeben werden. Kommen wir nun zur zweiten Zeile. Auch sie sollte übernommen werden. Der folgende Echo-Befehl benötigt nur Zeit und kann wegfallen. Der nächste Befehl ordnet den Speicherbereich, er wird übernommen. Wer keine zusätzlichen Erweiterungskarten in seinem Rechner hat, kann den BindDrivers-Befehl weglassen.

Der Wert des Addbuffers-Befehl könnte von 10 auf einen anderen Wert geändert werden, zum Beispiel 30, es muß aber nicht gemacht werden. Um eine schnellere Textausgabe zu haben, muß der FF-Befehl benutzt werden.

Als nächstes wird hier das T-Verzeichnis in die RAM-Disk verlegt. Dies muß nicht unbedingt sein. Das heißt, daß die beiden Befehle weggelassen werden können. Das gleiche gilt auch für die beiden nächsten Zeilen. Allerdings kann dann nicht mit den Befehlen Setenv und Getenv gearbeitet werden. Danach werden die verschiedenen Pfade gelegt. Sie sollten beibehalten werden. An der nächsten Zeile erkennen Sie, daß es sich hier um eine amerikanische Workbench-Version handelt. Die Zeile sollte auf *Sys:System/SetMap d* abgeändert werden. Das anschließende Laden der Workbench-Oberfläche sollte weiterhin beibehalten werden. Ob die Uhrzeit in der nächsten Zeile gestellt werden soll, hängt auch davon ab, ob Sie überhaupt eine batteriegepufferte Uhr in Ihrem Rechner haben. Der anschließende Resident-Befehl muß unbedingt übernommen werden, da ansonsten die neue Shell nicht funktioniert. Sie können dann selbst entscheiden, ob Sie den Execute-Befehl und eventu-

ell noch andere Befehle in die Residentliste aufnehmen oder nicht. Die neue Shell benötigt auch den neuen Console-Handler newcon:, deshalb muß auch die vorletzte Zeile beibehalten werden. Als letztes wird dann das CLI-Fenster geschlossen. Welche Zeilen Sie zusätzlich in die Sequence aufnehmen, bleibt Ihnen überlassen. Bedenken Sie aber, daß bei jedem erneuten Hochfahren des Rechners die gesamte Sequenz abgearbeitet wird, egal, wie lange es dauert. Es sind in dem S-Verzeichnis zwei weitere Startup-Dateien vorhanden, die sich schon vom Namen her sehr ähneln. Es sind dies die CLI-Startup- und die Shell-Startup-Dateien. Wie der Name verrät, ist die eine für das CLI und die andere für die Shell zuständig. Bei jedem Aufruf eines CLI- oder Shell-Fensters wird die entsprechende Datei abgearbeitet, das heißt, daß die Befehle, die in der Datei enthalten sind, zuerst ausgeführt werden. Dies ist beim CLI nur der Befehl *Prompt "%N> "*. Bei der Shell hingegen sind es folgende Befehle:

```
c:Prompt "%N.%S> "
alias xcopy copy [] clone
alias endshell endcli
```

Was der Alias-Befehl bewirkt und wie Sie ihn benutzen können, wird am Ende dieses Kapitels erläutert. Dort finden Sie auch noch weitere Anwendungsbeispiele zum Befehl Alias. Diese Beispiele können Sie auch in die Shell-Startup-Datei einbinden. Dazu müssen Sie nur mit einem Editor oder einer Textverarbeitung die Shell-Startup ergänzen. Die CLI-Startup-Datei können Sie in gleicher Weise ergänzen, allerdings dürfen Sie hier den Alias-Befehl nicht benutzen, denn dieser arbeitet nur mit der Shell zusammen. Ansonsten können Sie die gesamten CLI-Befehle verwenden.

6.3 Nützliche Batches

Mit der Benutzung von Batch-Dateien kann viel Tipparbeit und damit auch Zeit eingespart werden. Wir haben deshalb einige Batch-Dateien zusammengestellt, die Sie eventuell gebrauchen können. Falls Sie diese Dateien nicht benutzen, dann sehen Sie wenigstens einige Beispiele zu dem, was möglich ist. Vielleicht regt Sie das an, selbst Batch-Dateien zu erstellen. Wir haben zu-

erst einige Batches geschrieben, die mit den Befehlen der Workbench 1.2 auskommen. Diese funktionieren auch mit den Befehlen der Version 1.3, es folgen dann aber noch einige Batches, die nur mit den Befehlen der Workbench 1.3 erstellt werden können. Doch darauf wird an entsprechender Stelle noch einmal hingewiesen.

Wie Sie wissen, liegt diesem Buch eine Diskette bei. Auf dieser Diskette befindet sich ein Verzeichnis namens Batches. In diesem Verzeichnis finden Sie alle Batch-Dateien wieder, die Sie im weiteren Verlauf dieses Kapitels sehen werden. Die Namen, unter denen die einzelnen Dateien zu finden sind, finden Sie am Anfang jeder Datei als Kommentar eingefügt. Sie müssen sich also nicht die Mühe machen, die einzelnen Batches abzutippen, sondern Sie können diese verwenden.

6.3.1 Eine besondere Drucker-Batch-Datei

Haben Sie schon einmal eine Datei auf Ihren Drucker ausgedruckt? Dann haben Sie sicherlich festgestellt, daß der Text, der ausgedruckt wurde, immer die gleiche Gestaltung hatte. In diesem Abschnitt möchten wir Ihnen zeigen, wie Sie dies ändern können. Dazu sind allerdings einige Vorkenntnisse erforderlich. Diese möchten wir Ihnen ausführlich darlegen.

Wie Sie vielleicht wissen, kann man dem Drucker die verschiedensten Steuerzeichen übermitteln. Diese Steuerzeichen sind Escape-Sequenzen, die mit Hilfe des Befehls

```
COPY * TO PRT:
```

dem Drucker übermittelt werden. Nach Eingabe des Befehls und abschließendem Return werden alle Tastatureingaben direkt an den Drucker weitergeleitet. Sie bräuchten folglich die gewünschte Steuersequenz nur noch auf der Tastatur einzugeben. Da die direkte Übermittlung der Tastatureingabe an den Drucker innerhalb einer Batch-Datei nicht möglich ist, muß man hier einen Umweg gehen.

Diese Steuersequenzen leiten wir deshalb in verschiedene Dateien. Wir möchten Ihnen an einem Beispiel verdeutlichen, wie Sie dies erreichen. In unserem Beispiel behandeln wir die Steuersequenz für die Schriftart NLQ. Geben Sie nun folgende Zeile in CLI bzw. Shell ein:

```
COPY * TO NLQ
```

Nachdem Sie die Befehlszeile mit Return abgeschlossen haben, wird Ihr Laufwerk kurzzeitig anlaufen. Es richtet die Datei NLQ ein. Alle Tastatureingaben, die nun folgen, werden direkt in diese Datei geleitet. Das heißt, daß keine Ausgaben auf dem Bildschirm erfolgen werden. Sie tippen folglich "blind". Wenn Sie nun die eigentliche Steuersequenz für die Schriftart NLQ eingeben, dürfen keine Fehleingaben enthalten sein. Geben Sie nun unter besonderer Sorgfalt die folgenden Zeichen ein (mit ESC ist die speziell dafür vorgesehene ESC-Taste gemeint):

```
ESC[2"z
```

Wenn Sie nun die Return-Taste betätigen, wird Ihr Laufwerk wieder kurzzeitig anlaufen. Diesmal wird die obige Steuersequenz in die Datei geschrieben.

Um wieder in den "normalen" Modus zu gelangen, das heißt den Copy-Befehl aufzuheben, drücken Sie gleichzeitig die Tastenkombination Ctrl und \. Es erscheint nun wieder das gewohnte Prompt in CLI oder Shell. Damit ist die Übergabe der Steuersequenz in die Datei beendet.

Verfahren Sie für die Druckereinstellungen Fett (bold), Kursiv (italics) und Reset wie oben beschrieben. Benutzen Sie dazu die Dateinamen Fett, Kursiv und Reset an Stelle von NLQ. Welche Steuersequenz Sie für die einzelnen Druckereinstellungen benötigen, entnehmen Sie bitte dem Amiga-Benutzerhandbuch.

Bei der Übermittlung der Steuersequenzen in die Dateien kann es passieren, daß sich auch das Schriftbild im Normalmodus auf dem Bildschirm ändert. Das heißt, daß zum Beispiel nach Übermittlung der Steuersequenz kursiv auch die Ausgabe auf dem Bildschirm kursiv erscheint. Aufgrund dessen ist es ratsam, die

Steuersequenz für Reset als letztes zu übermitteln. Dadurch werden alle anderen Einstellungen aufgehoben.

Diese vier Dateien befinden sich auch auf der Diskette im Buch. Sie sind in dem Verzeichnis Druckersequenzen zu finden. Falls Sie noch weitere Steuersequenzen in Dateien kopieren, dann sollten Sie diese auch in dieses Unterverzeichnis kopieren.

Es folgt nun die Batch-Datei, die in der Überschrift erwähnt wurde. Geben Sie diese in einen Editor ein und speichern sie unter dem Namen Drucker ab, oder benutzen Sie die Datei Drucker im Verzeichnis Batches der Diskette im Buch.

```
.key Dateiname
;Drucker
if "<Dateiname>" eq ""
   echo "*nSie müssen eine Datei angeben.*n"
   quit
endif
if "<Dateiname>" eq "??"
   echo "*n*nAufruf: execute Drucker Dateiname*n"
   echo "Vergessen Sie nicht, die entsprechenden Pfade anzugeben.*n"
   quit
endif
if exists <Dateiname>
   ask "Möchten Sie die Datei in NLQ ausdrucken? (y/n) "
   if warn
      copy AmigaDOS-Buch:Druckersequenzen/Nlq to prt:
   else
      echo "Ok, kein NLQ."
   endif
   ask "Möchten Sie die Datei fett ausdrucken? (y/n) "
   if warn
      copy AmigaDOS-Buch:Druckersequenzen/Fett to prt:
   else
      echo "Ok, nicht fett."
   endif
   ask "Möchten Sie die Datei kursiv ausdrucken? (y/n) "
   if warn
      copy AmigaDOS-Buch:Druckersequenzen/Kursiv to prt:
   else
      echo "Ok, nicht kursiv."
   endif
   copy <Dateiname> to prt:
   copy AmigaDOS-Buch:Druckersequenzen/Reset to prt:
   echo "Ich bin fertig."
else
   echo "Ich konnte die Datei <Dateiname> nicht finden."
endif
```

Starten Sie diese Batch-Datei anschließend mit dem Befehl

```
EXECUTE AMIGADOS-BUCH:BATCHES/DRUCKER SYS:S/STARTUP-SEQUENCE
```

Daraufhin werden Ihnen nacheinander drei Fragen gestellt. Zum einen, ob Sie die Datei in NLQ ausdrucken wollen, zum anderen, ob Sie die Datei fett ausdrucken wollen und ob Sie die Datei kursiv ausdrucken wollen. Beachten Sie bitte bei der Beantwortung der Fragen, daß Ihr Drucker die entsprechenden Schrifteinstellungen verfügbar hat. Ist die dritte Frage beantwortet, beginnt kurz darauf der Drucker, die Datei Startup-Sequence aus dem S-Verzeichnis Ihrer Workbench-Diskette mit den entsprechenden Einstellungen auszudrucken. An Stelle der Startup-Sequence kann der gewünschte Dateiname eingesetzt werden. Wenn die Datei in einem Unterverzeichnis steht, dann ist darauf zu achten, daß der entsprechende Pfad mit angegeben wird.

In der obigen Batch-Datei sind nur die drei Beispieleinstellungen verwendet worden. Sie können nach Belieben auch die anderen zur Verfügung stehenden Druckereinstellungen in die Datei einbinden. Vorher müssen Sie allerdings die Escape-Sequenzen, wie oben beschrieben, in eine Datei schreiben.

6.3.2 Eigene Batch-Befehle erstellen

Wir zeigen Ihnen jetzt drei Beispiele, wie Sie eigene Befehle erstellen können. Es sind im eigentlichen Sinne keine Befehle, da sie nur in Verbindung mit dem Execute-Befehl zu verwenden sind, es sei denn, das s-Status-Bit der Batch-Datei ist gesetzt.

Da wäre zum einen der Backup-Befehl. Mit seiner Hilfe sind Sie in der Lage, von jeder Datei eine Kopie als Backup anzufertigen. Diese erzeugte Datei unterscheidet sich von der Originaldatei nur im Namen, er enthält die Endung .bak. Geben Sie nun folgende Zeilen in einen Editor ein, und speichern Sie sie unter dem Namen Backup ab, oder benutzen Sie die entsprechende Datei von der Diskette im Buch.

```
.key Dateiname
;Backup
if "<Dateiname>" eq ""
   echo "*nSie müssen eine Datei angeben.*n"
   quit
endif
if "<Dateiname>" eq "??"
   echo "*n*nAufruf: execute Backup Dateiname*n"
   echo "Vergessen Sie nicht, die entsprechenden Pfade anzugeben.*n"
   quit
endif
if exists <Dateiname>
   copy <Dateiname> to <Dateiname>.bak
else
   echo "*nIch konnte die Datei <Dateiname> nicht finden.*n"
endif
```

Sie können den Befehl anwenden, indem Sie ihn mit Execute und einem Dateinamen aufrufen. Eine komplette Befehlszeile sieht wie folgt aus:

```
EXECUTE AMIGADOS-BUCH:BATCHES/BACKUP Dateiname
```

Es wird eine Datei erstellt, die die Bezeichnung Dateiname.bak erhält. Der zweite Befehl lautet Window. Mit seiner Hilfe sind Sie in der Lage, bis maximal sechs CLI-Fenster mit nur einer eingegebenen Befehlszeile zu öffnen. Auch die folgenden Zeilen geben Sie in einen Editor ein. Diesmal speichern Sie das Eingegebene unter Window ab. Auch hier können Sie die Diskette im Buch verwenden.

```
.key Anzahl
;Window
if "<Anzahl>" eq "??"
   echo "*n*nAufruf: execute Window Anzahl*n"
   echo "Anzahl muß zwischen 1 und 6 einschließlich
liegen.*n"
   quit
endif
if "<Anzahl>" eq ""
   echo "*nSie müssen die Anzahl der Fenster eingeben.*n"
   quit
else
   skip <Anzahl>
   lab 6
    newcli "con:0/0/319/59/Ein_CLI"
   lab 5
    newcli "con:320/0/319/59/Ein_CLI"
   lab 4
```

```
    newcli "con:0/60/319/59/Ein_CLI"
   lab 3
    newcli "con:320/60/319/59/Ein_CLI"
   lab 2
    newcli "con:0/120/319/59/Ein_CLI"
   lab 1
    newcli "con:320/120/319/59/Ein_CLI"
endif
```

Aufrufen können Sie den Befehl durch Eingabe von

```
EXECUTE AMIGADOS-BUCH:BATCHES/WINDOW n
```

Dabei steht n für eine Zahl zwischen eins und sechs einschließlich. Wenn es zu einer Fehlermeldung kommt, kann es verschiedene Gründe dafür geben. Der Speicherplatz reicht nicht aus, oder die angegebene Zahl ist falsch gewählt.

Bevor wir den letzten Befehl ansprechen, geben wir Ihnen noch einen Hinweis. Er betrifft das Verzeichnis T. Es werden dort bei der Ausführung eines Execute-Befehls die verschiedensten Dateien abgelegt. Diese werden intern vom Rechner benutzt. Wenn Sie eine Batch-Datei aufrufen und die Meldung "Diskette ist schreibgeschützt" erscheint, dann erschrecken Sie nicht. Der Rechner muß nur auf das T-Verzeichnis zugreifen können.

Arbeiten Sie mit der RAM-Disk, dann tritt diese Meldung nicht auf. Es kann aber vorkommen, daß plötzlich ein Unterverzeichnis T auf der RAM-Disk erscheint, obwohl Sie es nicht angelegt haben. Dies hat der Rechner von sich aus gemacht.

Das letzte Thema, das wir in diesem Abschnitt behandeln, ist die RAM-Disk. Wir zeigen Ihnen eine Batch-Datei, die einige Befehle in die RAM-Disk kopiert. Dort stehen sie dem System in verkürzter Syntax zur Verfügung. Die Batch-Datei sieht wie folgt aus:

```
.key Parameter
;RAMon
if "<Parameter>" eq "??"
   echo "*n*nAufruf: execute RAMon*n"
   echo "Es brauchen keine Parameter angegeben zu werden.*n"
   quit
endif
if not "<Parameter>" eq ""
```

```
   echo "*nSie müssen keinen Parameter eingeben.*n"
endif
if not exists RAM:d
   makedir RAM:d
   copy sys:c/copy RAM:d/c
   copy sys:c/path RAM:d/p
   assign c: RAM:d
else
   copy sys:c/copy RAM:d/c
   c sys:c/path RAM:d/p
endif
p add sys:c
c sys/c:dir RAM:d/d
c sys/c:execute RAM:d/ex
c sys/c:delete RAM:d/del
c sys/c:type RAM:d/t
c sys/c:rename RAM:d/r
c sys/c:echo RAM:d/e
e "*nDie verkürzten Befehle sind nun verfügbar.*n"
```

Geben Sie die Zeilen in einen Editor ein, und speichern Sie diese unter der Bezeichnung RAMon ab, oder benutzen Sie die entsprechende Datei aus dem Verzeichnis Batches der Diskette im Buch. Nachdem Sie die Datei mit *Execute AmigaDOS-Buch:Batches/RAMon* abgearbeitet haben, stehen Ihnen die verkürzten Befehle C, P, D, EX, Del, T, R und E zur Verfügung. Welche Befehle diese Buchstaben repräsentieren, entnehmen Sie aus den obigen Zeilen. Dem Kopieren anderer bzw. weiterer Befehle steht nichts im Wege. Sie müssen jedoch beim Verkürzen der Befehle darauf achten, daß keine Abkürzung mehr als einmal benutzt wird.

Die Zurverfügungstellung der verkürzten Befehle kann auch in die Startup-Sequence eingebunden werden. Dies sollte erst am Ende der Sequence geschehen.

Um den durch die verkürzten Befehle in Anspruch genommenen Speicherplatz wieder frei zu bekommen, haben wir die folgende Batch-Datei erstellt:

```
.key Parameter
;RAMoff
if "<Parameter>" eq "??"
   echo "*n*nAufruf: execute RAMoff*n"
   echo "Es brauchen keine Parameter angegeben zu werden.*n"
   quit
endif
```

```
if not "<Parameter>" eq ""
   echo "*nSie müssen keine Parameter eingeben.*n"
endif
if exists RAM:t
   delete RAM:t all quiet
endif
if exists ram:d
   cd ram:d
   sys:c/delete #? quiet
   sys:c/cd sys:
   echo "*nDie verkürzten Befehle sind nicht mehr verfügbar.*n"
endif
```

Diese Zeilen sollten Sie unter dem Namen RAMoff abspeichern oder die entsprechende Datei auf der Diskette im Buch benutzen. Durch Eingeben der Befehlszeile *Execute AmigaDOS-Buch:Batches/RAMoff* löschen Sie sowohl die verkürzten Befehle als auch das eventuell vorhandene Unterverzeichnis T in der RAM-Disk. Der dadurch frei werdende Speicherplatz kann anschließend für andere Zwecke genutzt werden.

Wenn Sie jetzt einwenden, daß das Verzeichnis d in der RAM-Disk nicht gelöscht ist, so müssen wir Ihnen recht geben. Allerdings ist das Verzeichnis leer. Es befinden sich keine Befehle mehr darin. Warum wir das Verzeichnis nicht gelöscht haben, hat einen speziellen Grund. Der Amiga würde uns bei dem Versuch, dieses Verzeichnis zu löschen, eine Fehlermeldung auf den Bildschirm bringen, die letztendlich besagt, daß dieses Verzeichnis nicht gelöscht werden kann. Dies liegt darin begründet, daß wir innerhalb der RAMon-Batch-Datei mit dem Assign-Befehl den Zugriff auf das D-Verzeichnis in der RAM-Disk festgelegt haben. Um diesen Zugriff weiterhin zu gewährleisten, muß folglich das Verzeichnis vorhanden bleiben.

Dieser kleine Nachteil wird durch den großen Vorteil, daß Sie sowohl die verkürzten als auch alle originalen CLI-Befehle benutzen können, aufgehoben.

6.3.3 Die Arbeits-Workbench 1.3

Im weiteren Verlauf dieses Kapitels werden die Befehle der Workbench 1.3 benötigt. Sie können zwar versuchen, die Batches

mit den 1.2-Befehlen zu erstellen, doch selbst wenn alle benutzten Befehle auch auf der Workbench 1.2 vorhanden sind, heißt das nicht, daß es funktionieren muß. Die meisten 1.3-Befehle heißen zwar genauso, sie wurden aber größtenteils um einige Optionen erweitert.

Wie die Überschrift schon sagt, möchten wir in diesem Unterkapitel eine Batch-Datei zeigen, die aus der originalen Workbench 1.3 eine Arbeits-Workbench 1.3 erstellt. Was wir unter Arbeits-Workbench verstehen, sei kurz erklärt.

Wenn Sie sich an Ihren Computer setzen, dann wissen Sie im allgemeinen, was Sie tun wollen. Dazu werden Sie in den meisten Fällen nur einen kleinen Teil der auf der Workbench enthaltenen Befehle oder Programme benutzen. Es wird allerdings öfter vorkommen, daß Sie noch einige zusätzliche Programme, sei es ein Diskettenmonitor oder einen anderen Editor oder etwas ähnliches, benötigen. Deshalb empfiehlt es sich, mehrere Arbeits-Workbenches anzufertigen, die alle einen mehr oder weniger gleichen Grundaufbau haben, dann aber die verschiedensten Programme zusätzlich beinhalten. Die eine hat zum Beispiel eine Textverarbeitung als Hauptprogramm mit auf der Diskette, die andere eine Dateiverwaltung usw. Die Batch-Datei, die wir Ihnen zeigen, hat folgende Funktion: Sie fertigt eine Arbeitsdiskette an, auf der alle grundlegenden Dinge vorhanden sind.

Doch bevor wir uns mit der Datei befassen, sollten Sie eine Kopie der originalen Workbench 1.3 anfertigen. Anschließend stellen Sie auf der Kopie mittels Preferences alles nach Ihren Wünschen ein und speichern es ab. Dies ist sehr wichtig, da auf der späteren Arbeitsdiskette keine Möglichkeit mehr besteht, die Einstellungen zu ändern. Auch die Startup-Sequence und die Mount-List sollten Sie vor der Benutzung der Batch-Datei auf Ihre Bedürfnisse abändern. Die originale Workbench 1.3 haben Sie doch wieder an ihren sicheren Ort gelegt, oder nicht? Gut. Es folgt jetzt der Inhalt der Batch-Datei versehen mit Zeilennummern, das heißt, daß Sie beim Abtippen die jeweils vor den Befehlen stehenden Zahlen nicht mit eingeben dürfen. Wir haben dies getan, damit bei der anschließenden Erläuterung die einzelnen Zeilen genau angegeben werden können.

Wir haben dies getan, damit bei der anschließenden Erläuterung die einzelnen Zeilen genau angegeben werden können.

```
 0  ;Arbeitsdisk
 1  sys:c/makedir ram:Arbeitsdisk
 2  sys:c/makedir ram:Arbeitsdisk/c
 3  sys:c/copy sys:c/Copy ram:Arbeitsdisk/c
 4  sys:c/cd ram:Arbeitsdisk/c
 5  copy sys:c/Addbuffers ram:Arbeitsdisk/c
 6  copy sys:c/Ask ram:Arbeitsdisk/c
 7  copy sys:c/Assign ram:Arbeitsdisk/c
 8  copy sys:c/Binddrivers ram:Arbeitsdisk/c
 9  copy sys:c/CD ram:Arbeitsdisk/c
10  copy sys:c/Delete ram:Arbeitsdisk/c
11  copy sys:c/Dir ram:Arbeitsdisk/c
12  copy sys:c/Echo ram:Arbeitsdisk/c
13  copy sys:c/Endcli ram:Arbeitsdisk/c
14  copy sys:c/FF ram:Arbeitsdisk/c
15  copy sys:c/If ram:Arbeitsdisk/c
16  copy sys:c/Install ram:Arbeitsdisk/c
17  copy sys:c/Lab ram:Arbeitsdisk/c
18  copy sys:c/Loadwb ram:Arbeitsdisk/c
19  copy sys:c/Makedir ram:Arbeitsdisk/c
20  copy sys:c/Mount ram:Arbeitsdisk/c
21  copy sys:c/Newshell ram:Arbeitsdisk/c
22  copy sys:c/Path ram:Arbeitsdisk/c
23  copy sys:c/Prompt ram:Arbeitsdisk/c
24  copy sys:c/Resident ram:Arbeitsdisk/c
25  copy sys:c/Run ram:Arbeitsdisk/c
26  copy sys:c/Setalert ram:Arbeitsdisk/c
27  copy sys:c/Setclock ram:Arbeitsdisk/c
28  copy sys:c/Skip ram:Arbeitsdisk/c
29  copy sys:c/Type ram:Arbeitsdisk/c
30  copy sys:c/Xicon ram:Arbeitsdisk/c
31  ;Hier können weitere Befehle folgen
32  copy sys:.info ram:Arbeitsdisk
33  copy sys:Clock#? ram:Arbeitsdisk
34  copy sys:Disk.info ram:Arbeitsdisk
35  copy sys:Shell#? ram:Arbeitsdisk
36  copy sys:System.info ram:Arbeitsdisk
37  makedir ram:Arbeitsdisk/System
38  copy sys:System ram:Arbeitsdisk/System all
39  makedir ram:Arbeitsdisk/L
40  copy sys:L ram:Arbeitsdisk/L all
41  makedir ram:Arbeitsdisk/Devs
42  copy sys:Devs/#? ram:Arbeitsdisk/Devs
43  makedir ram:Arbeitsdisk/Devs/Keymaps
44  copy sys:Devs/Keymaps/d ram:Arbeitsdisk/Devs/Keymaps
45  copy sys:Devs/Keymaps/usa1 ram:Arbeitsdisk/Devs/Keymaps
46  makedir ram:Arbeitsdisk/Devs/Printers
47  copy sys:Devs/Printers/EpsonQ
    ram:Arbeitsdisk/Devs/Printers
48  makedir ram:Arbeitsdisk/S
```

```
53  copy sys:Fonts/Topaz.font ram:Arbeitsdisk/Fonts
54  makedir ram:Arbeitsdisk/Fonts/Topaz
55  copy sys:Fonts/Topaz/11 ram:Arbeitsdisk/Fonts/Topaz
56  makedir ram:Arbeitsdisk/Libs
57  copy sys:Libs ram:Arbeitsdisk/Libs all
58  makedir ram:Arbeitsdisk/Utilities
59  echo "*n*nLegen Sie bitte eine Diskette ins Laufwerk df0:."
60  echo "Diese Diskette wird dann formatiert und anschließend"
61  echo "zur neuen Arbeits-Workbench.*n"
62  ram:Arbeitsdisk/System/format drive df0: name
    "Arbeits-Workbench 1.3" noicons
63  install df0:
64  copy ram:Arbeitsdisk df0: all
65  cd "Arbeits-Workbench 1.3"
66  c/delete ram:Arbeitsdisk all quiet
67  c/echo "Fertig!"
```

Diese Batch-Datei unterscheidet sich von den bisher besprochenen Dateien in einem Punkt. Am Anfang wird die Befehlseingabe weder auf *??* noch auf eventuell eingegebene Parameter hin abgefragt. Dies hat einen ganz besonderen Grund. Die Datei Arbeitsdisk im Verzeichnis Batches auf der Diskette im Buch soll mit Hilfe des XIcon-Befehls der Workbench 1.3 auch mit der Maus aktiviert werden können. Die Benutzung des XIcon-Befehls hat allerdings den Nachteil, daß nur die Befehle in einer Batch-Datei benutzt werden dürfen, die auch in CLI bzw. Shell eingegeben werden dürfen. Und dazu zählen die Befehle If, Else usw. eben nicht. Was die einzelnen Zeilen in dieser Datei bewirken, wissen Sie wahrscheinlich schon. Falls aber noch Unklarheiten bei der ein oder anderen Zeile auftreten, dann können Sie diese durch die folgende Erläuterung beseitigen.

Zeile 1-2

Es wird in der RAM-Disk ein Verzeichnis Arbeitsdisk und in diesem wiederum ein Verzeichnis c eingerichtet.

Zeile 3-4

In das oben eingerichtete Verzeichnis c wird der Befehl Copy kopiert, und anschließend wird in dieses Verzeichnis gewechselt. Dies hat den Vorteil, daß die nachfolgenden Befehle nicht erst von der Diskette geladen werden müssen.

Dies hat den Vorteil, daß die nachfolgenden Befehle nicht erst von der Diskette geladen werden müssen.

Zeile 5-30

Jede einzelne Zeile kopiert einen Befehl aus dem c-Verzeichnis der Workbench in das c-Verzeichnis der RAM-Disk. Warum so viele, werden Sie fragen. Die große Anzahl der Befehle liegt darin begründet, daß alle kopierten Befehle entweder in der Startup-Sequence oder innerhalb dieser Batch-Datei verwendet werden. Weiterhin sind einige Befehle kopiert worden, die so oft gebraucht werden, daß sie in dieser Liste nicht fehlen dürfen. Dies sind zum Beispiel der Dir- oder der Type-Befehl.

Zeile 31

Diese Zeile ist nur ein Kommentar und soll Sie darauf aufmerksam machen, daß hier die geeignete Stelle ist, um eventuell noch weitere Befehle zu kopieren, die Sie benötigen.

Zeile 32-36

Hier werden alle benötigten Dateien aus dem Hauptverzeichnis der Workbench in das Arbeitsdisk-Verzeichnis der RAM-Disk kopiert.

Zeile 37-40

Die Verzeichnisse System und L werden innerhalb des Arbeitsdisk-Verzeichnisses eingerichtet. Danach werden jeweils die entsprechenden Inhalte (d.h. Dateien) der gleichnamigen Verzeichnisse der Workbench in diese kopiert.

Zeile 41-42

Das Verzeichnis Devs wird eingerichtet, und alle Dateien (nicht die Unterverzeichnisse) aus diesem Verzeichnis werden von der Workbench in die RAM-Disk kopiert. Unter diesen Dateien befindet sich auch eine Datei namens System-Configuration. In ihr sind alle Informationen enthalten, die Sie in Preferences eingestellt haben. Die Einstellungen werden beim Einschalten des

Rechners berücksichtigt, können aber nicht verändert werden, da das Programm Preferences nicht mit kopiert wird. Eine weitere Datei hat den Namen Mountlist. Auch diese sollten Sie nach Ihren Wünschen vor Benutzung dieser Batch-Datei abändern.

Zeile 43-45

Im Verzeichnis Devs wird das Verzeichnis Keymaps eingerichtet. Danach werden die Keymaps d (für die deutsche Tastaturbelegung) und usa1 (für die amerikanische Tastaturbelegung) von der Workbench in die RAM-Disk kopiert.

Zeile 46-47

Hier geschieht das gleiche wie in Zeile 43-45, allerdings mit dem Verzeichnis Printers und dem Druchertreiber EpsonQ. Sie müssen hier den Druckertreiber kopieren, den Sie auf der Workbench mittels Preferences ausgewählt haben. Die Datei EpsonQ dient hier nur als Beispiel.

Zeile 48-50

Das Verzeichnis S wird auf der RAM-Disk eingerichtet. Anschließend werden die Startup-Sequence und die CLI- und Shell-Startup-Dateien in das Verzeichnis kopiert.

Zeile 51

Für die temporären Dateien wird ein Verzeichnis T eingerichtet.

Zeile 52-55

Diese Zeilen zeigen, wie ein Font von der Workbench übernommen wird. Sollten Sie weitere Fonts übernehmen wollen, dann verfahren Sie wie in diesen Zeilen gezeigt.

Zeile 56-57

Es wird das Verzeichnis Libs eingerichtet. Danach wird der gesamte Inhalt des Verzeichnisses der Workbench in das der RAM-Disk kopiert.

Zeile 58

Da in den meisten Fällen in der Startup-Sequence ein Pfad zu diesem Verzeichnis gelegt wird, haben wir hier das dazu benötigte Verzeichnis eingerichtet. Wie Sie sehen haben wir keine Dateien in dieses Verzeichnis kopiert. Falls Sie jedoch öfter mit dem Calculator, dem Notepad oder einem anderen Programm arbeiten, dann kopieren Sie die entsprechenden Programme in dieses Unterverzeichnis.

Zeile 59-61

Mittels dieser Zeilen wird Text auf dem Bildschirm ausgegeben.

Zeile 62-63

Nachdem Sie die Workbench aus dem Laufwerk df0: entfernt und eine leere Diskette dort hineingelegt haben, wird die Diskette formatiert und anschließend installiert (d.h. bootfähig gemacht).

Zeile 64

In dieser Zeile wird die neu formatierte Diskette zur Arbeits-Workbench gemacht, denn es wird das gesamte Verzeichnis Arbeitsdisk der RAM-Disk mit deren Unterverzeichnissen und Dateien auf die neue Diskette übertragen.

Zeile 65-66

Es wird auf die Arbeits-Workbench gewechselt, und die RAM-Disk wird wieder geleert.

Zeile 67

Zum Schluß erfolgt eine Fertigmeldung auf dem Bildschirm.

Einige Anmerkungen zur Batch-Datei

Sie benötigen zur Durchführung der Datei nur ein Laufwerk. Da der benötigte Speicherplatz recht groß ist, sollten Sie über mindestens ein Megabyte RAM verfügen. Haben Sie diesen Speicherplatz nicht zur Verfügung, können Sie sich helfen, indem Sie einige Befehle aus dem Verzeichnis c nicht innerhalb der Batch-Datei kopieren. Sie können alle Befehle, die nicht innerhalb der Batch-Datei benutzt werden, erst einmal weglassen. Diese müssen Sie anschließend, nachdem die Batch-Datei fertig ist, nachträglich kopieren. Der dadurch eingesparte Platz müßte ausreichen, um auch mit 512 KByte auszukommen.

Wenn Sie diese Batch-Datei benutzen, dann müssen Sie vorher Ihren Amiga mit der Kopie der Workbench 1.3 booten. Anschließend können Sie die Datei wie oben schon beschrieben starten.

Haben Sie mit dem Batch eine Arbeitsdiskette hergestellt, dann können Sie die Programme, die Sie noch gebrauchen, auf diese Diskette kopieren. Beachten Sie aber, daß diese Programme eventuell noch andere Dateien benötigen, die zur Zeit nicht auf der Diskette vorhanden sind. Dies sind zum Beispiel zusätzliche Befehle oder Fonts. Auch diese Dateien müssen Sie dann auf Ihre Arbeits-Workbench in die entsprechenden Verzeichnisse kopieren.

6.3.4 Starten von Batch-Dateien mit der Maus

Haben Sie sich auch schon mal darüber geärgert, daß Sie bei der Benutzung von Batch-Dateien sehr viel Vorarbeit leisten mußten, bevor Sie die Datei starten konnten? Zuerst muß das Diskettensymbol doppelt angeklickt werden, dann muß die richtige Schublade geöffnet werden, um dann zuerst einmal ein CLI öffnen zu können. Erst danach konnte die Batch-Datei mit einiger Tipparbeit gestartet werden. Mit der Workbench 1.3 wird ein Befehl geliefert, mit dessen Hilfe Sie sich einen Teil dieser Arbeit ersparen können. Dieser Befehl heißt XIcon und ist im c-Verzeichnis zu finden. Wie wird der Befehl genutzt? Diese Frage möchten wir an einem Beispiel erläutern. Erstellen Sie dafür ein

Batch in der RAM-Disk. Geben Sie dazu folgende Zeile in die Shell ein:

```
echo >ram:Batch "dir df0:*ncd ram:*ntype Batch"
```

Mit *dir Ram:* können Sie nachschauen, ob sich die Datei Batch in der RAM-Disk befindet. Ausführen können Sie die Datei, indem Sie execute ram:batch in die Shell eingeben. Es wird daraufhin das Inhaltsverzeichnis der Diskette, die sich in Laufwerk df0: befindet auf dem Bildschirm sichtbar, gefolgt von drei Befehlszeilen, die den Inhalt der Datei Batch darstellen. Doch was muß getan werden, damit diese Datei auch mit der Maus gestartet werden kann? Als erstes benötigen wir ein Icon. Dieses Icon muß ein Project-Icon sein, wie zum Beispiel das Shell-Icon. Wir werden dieses in unserem Beispiel benutzen. Wenn Sie ein anderes Icon nehmen möchten, dann können Sie ganz einfach prüfen, ob es sich um ein Tool-Icon handelt. Klicken Sie das Icon einmal an und halten anschließend die rechte Maustaste gedrückt. Sofort erscheint in der obersten Zeile des Bildschirms eine Menüleiste. Aktivieren Sie den Menüpunkt Info in der Workbench-Spalte. Es erscheint daraufhin ein Info-Fenster. In der linken oberen Ecke dieses Fensters befindet sich das Wort Type. Direkt daneben muß sich das Wort Tool befinden. Steht dort zum Beispiel Project oder Drawer, dann können Sie dieses Icon für unseren Zweck nicht benutzen. Auf der Diskette im Buch befindet sich im Verzeichnis Batches die Datei ramcontrol. Zu dieser existiert ein Icon, also auch ein ramcontrol.info-File. Dieses Icon werden wir in unserem Beispiel benutzen. Kopieren Sie das Icon in die RAM-Disk und geben Sie der Datei sofort den Namen der Beispiel-Batch-Datei, wobei die Endung .info beibehalten werden muß. Dies geschieht mit dem Befehl

```
copy AmigaDOS-Buch:Batches/ramcontrol.info ram:Batch.info
```

Wenn Sie eine andere .info-Datei benutzen möchten, dann müssen Sie dementsprechend einen anderen Quellnamen bzw. Quellpfad eingeben.

Haben Sie den Befehl eingegeben, dann klicken Sie das RAM-Disk-Symbol doppelt an. In dem erscheinenden Fenster erkennen Sie ein Icon, mit dem Namen unserer Batch-Datei. Klicken Sie

es einmal mit der linken Maustaste an. Danach aktivieren Sie den Menüpunkt Info, wie oben schon beschrieben. Es erscheint das Info-Fenster.

Innerhalb dieses Fensters befindet sich ein Kasten, worin der Text DEFAULT TOOL steht. Falls Sie mit dem ramcontrol-Icon arbeiten, dann steht rechts daneben *SYS:C/Xicon*. Haben Sie ein anderes Icon gewählt, dann wird dort auch ein anderer Text stehen. Ändern Sie diesen Text um, so daß Sie auch *SYS:C/Xicon* dort stehen haben. Klicken Sie dazu den Text an, und löschen Sie diesen mit der Del-Taste. Anschließend geben Sie den neuen Text ein. Dieser Text gibt den Pfad an, in dem sich der Befehl XIcon befindet. Haben Sie diesen Text eingegeben, dann schließen Sie das Fenster wieder, indem Sie das Feld SAVE anklicken. Sofort verschwindet das Fenster vom Bildschirm. Wenn Sie das RAM-Disk-Fenster nicht wieder geschlossen haben, dann sehen Sie das Batch-Icon in diesem Fenster. Klicken Sie es doppelt an. Es öffnet sich daraufhin ein weiteres Fenster. In diesem Fenster erscheinen die gleichen Ausgaben, wie es oben der Fall war, als die Batch-Datei mit *execute* gestartet wurde. Nachdem alle Ausgaben in dem Fenster ausgegeben wurden, bleibt das Fenster kurze Zeit geöffnet, bevor es dann automatisch geschlossen wird. Falls Sie die Zeit, die das Fenster noch göffnet bleibt, als zu kurz empfinden, erweitern Sie die Batch-Datei um den Befehl Wait 30. Das Fenster wird dann 30 Sekunden länger göffnet bleiben.

So, das war ein Beispiel, wie Sie eine Batch-Datei von der Workbench aus starten können. Sie können so mit allen Batch-Dateien verfahren, mit einer Einschränkung. Die Batch-Dateien dürfen nur solche Befehle enthalten, die auch in CLI oder Shell eingegeben werden dürfen. Befehle wie Skip, Lab, If usw. sind also nicht erlaubt. Sie können dabei natürlich auch mit den Laufwerken arbeiten. Der XIcon-Befehl ist nicht nur für die RAM-Disk bestimmt. Falls Ihnen das Icon nicht gefallen sollte, dann können Sie es mit dem Icon-Editor verändern.

6.3.5 Batch-Dateien "typen"

Falls Sie schon früher, bevor Sie dieses Buch durchgearbeitet haben, Batch-Dateien erstellt haben, dann haben Sie diese hoffentlich in das Unterverzeichnis s der Workbench kopiert. Dort schaut der Amiga selbständig nach, ob die Batch-Datei vorhanden ist, die Sie starten wollen, wenn er sie im Hauptverzeichnis nicht gefunden hat. Dies nur als kleinen Hinweis zu der nun folgenden Batch-Datei. Sie gibt den Inhalt jeder einzelnen Datei aus einem Verzeichnis aus, welches Sie angeben können. Sie sollten aber darauf achten, daß in dem angegebenen Verzeichnis hauptsächlich Textdateien wie zum Beispiel Batch-Dateien oder Source-Dateien stehen, da das "Typen" von zum Beispiel compilierten Programmen keinen Sinn ergibt.

```
.key Verzeichnis
;Typen
if "<Verzeichnis>" eq "??"
   echo "*n*nAufruf: execute Typen Verzeichnis*n"
   echo "Vergessen Sie nicht, die entspr. Pfade anzugeben.*n"
   quit
endif
if "<Verzeichnis>" eq ""
   echo "*n*nSie müssen ein Verzeichnis angeben.*n"
   quit
endif
if exists "<Verzeichnis>"
   cd "<Verzeichnis>"
else
   echo "*nIch kann das Verzeichnis <Verzeichnis> nicht finden.*n"
   quit
endif
list >ram:Type.bat #? lformat="type %s"
execute ram:Type.bat
delete ram:Type.bat
cd sys:
```

Sie starten diese Datei mit *execute*, oder Sie setzen das S-Bit dieser Datei. Beachten Sie, daß Sie das Laufwerk und das Verzeichnis angeben müssen. Da die Datei auf der Diskette im Buch Typen heißt, könnte ein Aufruf wie folgt aussehen:

```
execute AmigaDOS-Buch:Batches/Typen sys:s
```

Die anschließende Ausgabe können Sie anhalten, indem Sie die rechte Maustaste gedrückt halten. Die Ausgabe kann auch auf

einen Drucker erfolgen. Dazu muß die lformat-Option nur in *lformat="type >prt: %s"* geändert werden.

6.3.6 Alles in die RAM-Disk

Die in diesem Unterkapitel gezeigte Batch-Datei sollten Sie nur verwenden, wenn Sie eine Speicher-Erweiterung in Ihrem Rechner haben, denn wie die Kapitelüberschrift schon sagt, wird "Alles" in die RAM-Disk kopiert. Was ist alles? Unter "Alles" verstehen wir die gesamte Kontrolle, die normalerweise die Workbench hat. Das heißt, wir kopieren die gesamte Workbench in die RAM-Disk und legen alle Zugriffe, die sonst auf die Workbench gehen, auf die RAM-Disk um. Sie können natürlich auch die resetfeste RAM-Disk dazu benutzen, dazu müssen Sie eventuell in der Mount-List die Parameter der resetfesten RAM-Disk ändern. Sie müssen den Parameter HighCyl so groß wählen, daß der gesamte Inhalt der Workbench in der resetfesten RAM-Disk Platz hat. Bedenken Sie bei der Benutzung dieser Dateien, daß sehr viel Speicherplatz benötigt wird, der den anderen Anwendungen nicht mehr zur Verfügung steht. Die erste der nun folgenden Batch-Dateien benutzt die normale RAM-Disk. Die zweite dagegen benutzt die resetfeste RAM-Disk.

```
;Ramcontrol
echo "*nDie Verzeichnisse werden kopiert.*n"
copy sys: ram: all
cd ram:
echo "*nDie Kontrolle wird an die RAM-Disk übergeben.*n"
assign sys: ram:
assign c: sys:c
assign l: sys:l
assign fonts: sys:fonts
assign s: sys:s
assign devs: sys:devs
assign libs: sys:libs
assign t: sys:t
assign utilities: sys:utilities
assign prefs: sys:prefs
assign system: sys:system
assign empty: sys:empty
echo "*nFertig.*n"


;Radcontrol
if exists rad:Flag
```

```
   skip L1
else
   echo "*nDie Verzeichnisse werden kopiert.*n"
   copy sys: rad: all
endif
echo >rad:Flag "Flag gesetzt."
lab L1
cd rad:
echo "*nDie Kontrolle wird an resetfeste RAM-Disk übergeben.*n"
assign sys: rad:
assign c: sys:c
assign l: sys:l
assign fonts: sys:fonts
assign s: sys:s
assign devs: sys:devs
assign libs: sys:libs
assign t: sys:t
assign utilities: sys:utilities
assign prefs: sys:prefs
assign system: sys:system
assign empty: sys:empty
echo "*nFertig.*n"
```

Sie sehen, daß die zweite Datei etwas umfangreicher ist. Das hat den Grund, daß nach einem Warmstart (Amiga Amiga Ctrl) der Inhalt der resetfesten RAM-Disk, sobald sie wieder "gemountet" wird, immer noch vorhanden ist. Die Dateien der Workbench müssen also nicht noch einmal kopiert werden. In den zusätzlichen Zeilen gegenüber der Ramcontrol-Datei wird dies berücksichtigt.

6.4 Neue Shell-Befehle mit Alias

Die Version 1.3 der Workbench hat ein sehr leistungsfähiges Programm. Es ist die Shell. Die Vorteile gegenüber dem CLI sind sehr groß. Zum einen können die Befehlszeilen editiert werden, und zum andern werden die eingegebenen Befehlszeilen der Reihe nach in einem Puffer gespeichert. Doch es gibt noch einen großen Vorteil. Dieser besteht in dem Befehl Alias. Mit ihm sind wir in der Lage, die vorhandenen CLI-Befehle in einer anderen Weise zu nutzen. Warum wird das hier im Kapitel Batches erläutert, werden Sie fragen. Das hat den Grund, daß der Alias-Befehl fast die gleiche Funktion hat wie eine Batch-Datei. Allerdings bestehen diese Batch-Dateien aus nur einer Zeile. Die Zeile beginnt mit dem Befehlswort Alias. Danach folgt eine

Zeichenkette, die den Namen angibt, mittels dem die Batch-Datei angesprochen wird. Zum Schluß kommt dann noch der Befehl, der die Batch-Datei bildet. Das hört sich sehr schwierig an, ist es aber nicht. Bevor wir das an einem Beispiel verdeutlichen, noch ein Hinweis. Alle in diesem Kapitel erstellten Alias-Befehle haben wir in eine Batch-Datei namens Alias.Bat auf der Diskette im Buch geschrieben. Diese befindet sich auch im Unterverzeichnis Batches. Wenn Sie ein Shell-Fenster geöffnet haben, müssen Sie *execute AmigaDOS-Buch:Batches/Alias.Bat* eingeben, und es stehen Ihnen alle folgenden Alias-Befehle zur Verfügung.

Zurück zu unserem Beispiel. Geben Sie die folgende Zeile in die Shell ein:

```
Alias Ramdir Dir Ram:
```

Anschließend steht Ihnen der neue Befehl Ramdir zur Verfügung. Sie sehen, es ist ganz einfach. Zuerst das Befehlswort (*Alias*), dann die Bezeichnung des neuen Befehls (*Ramdir*), und danach steht das, was der Befehl bewirkt (*Dir Ram:*). Das von uns gewählte Beispiel ist nicht sehr sinnvoll, aber als Beispiel reicht es aus. Welche Alias-Befehle Sie zur Zeit benutzen können und wie sie aufgebaut sind, erfahren Sie, indem Sie nur *Alias* gefolgt von RETURN eingeben. Um Ihnen einige Anregungen zu dem Alias-Befehl zu geben, haben wir einige neue Befehle erstellt. Sie können diese Zeilen übrigens auch in die Shell-Startup-Datei einbinden, dann stehen Ihnen diese Befehle in jedem Shell-Fenster zur Verfügung. Oder Sie schreiben sie in eine Batch-Datei, die Sie bei Bedarf einmal aufrufen müssen, um die Befehle nutzen zu können.

1. Sie möchten das Laufwerk wechseln, möchten aber nicht jedesmal soviel tippen. Benutzen Sie folgende Zeilen:

```
alias 0 cd df0:
alias 1 cd df1:
alias 2 cd df2:
alias r cd ram:
      .
      .
      .
```

Nach Eingabe dieser Zeilen sind Sie in der Lage, das Laufwerk zu wechseln indem Sie 0 oder 1 oder 2 oder r eingeben. Sie können natürlich auch mit anderen Laufwerken so verfahren (z.B. dh0:). Des weiteren können Sie auch gleich in ein Verzeichnis in dem angegebenen Laufwerk wechseln. Möchten Sie in das Verzeichnis c des Laufwerkes df1: wechseln, müssen Sie nur den Befehl *1 c* eingeben.

2. Es soll der Inhalt eines Shell-Fensters gelöscht werden.

```
Alias CLS echo "*ec"
```

Geben Sie die Zeile in die Shell ein. Danach geben Sie CLS ein. Der Inhalt der Shell wird gelöscht, und das Prompt meldet sich oben im Fenster. Dies wird dadurch erreicht, daß mit dem Echo-Befehl eine Escape-Sequenz ausgegeben wird. Diese Sequenz steht innerhalb der Anführungszeichen. Die ersten beiden Zeichen teilen dem Rechner mit, daß nun eine Escape-Sequenz folgt. Das c bewirkt dann das Löschen des Bildschirms. Im Anhang A finden Sie weitere Escape-Sequenzen, die Sie ebenfalls benutzen können. Das Prompt steht aber nicht ganz oben, sondern in der zweiten Zeile. Um es nach dem Löschen in die ersten Zeile zu bekommen, müssen wir ein wenig tricksen. Zuerst müssen wir das Prompt-Zeichen mit folgender Zeile ein wenig abändern:

```
Prompt "*e[11y*e[33m*e[1m*e[3m%s*e[0m*n*e[t"
```

Haben Sie die Zeile eingegeben und mit Return abgeschlossen, dann sehen Sie schon die Auswirkungen. Das Verzeichnis, in dem Sie sich zur Zeit befinden, ist kursiv und in oranger Farbe dargestellt. Außerdem erfolgt die Eingabe erst in der nächsten Zeile. Diese Einstellungen bewirken die verschiedenen Escape-Sequenzen im Prompt-Befehl.

Geben Sie nun den neuen CLS-Befehl ein.

```
Alias CLSC echo "*ec*e[2y*e[2t"
```

Wenn Sie nun den Bildschirm mit CLSC löschen, dann sehen Sie, daß die Verzeichnisangabe in der obersten Zeile des Fensters steht.

3. Da wir die Escape-Sequenzen im vorherigen Befehl schon angesprochen haben, möchten wir Ihnen einige weitere Anwendungen in dieser Art zeigen. Die ersten Beispiele benutzen zwar den Alias-Befehl nicht, sind aber dennoch interessant.

```
echo "*e[1m*e[3mDies ist fett und kursiv.*e[0m"
echo "*e[32m*e[43mDies ist schwarze Schrift auf orange-
                farbigem Untergrund.*e[0m"
echo "*e[7m*e[4mDies ist invertiert und unterstrichen.*e[0m"
```

   Sie sehen, die Änderungen gelten nur für die eine Ausgabe. Bleibende Änderungen können auch den Alias-Befehl beinhalten.

```
Alias Prom1 Prompt "*e[32m*e[43m%s> "
Alias Prom2 Prompt "*e[42m*e[31m%s> "
Alias Prom3 Prompt "*e[41m*e[33m%s> "
Alias Prom4 Prompt "%nter Task*n%s*n- "
                .
                .
                .
```

   In den normalen Modus kommen Sie, indem Sie *echo "*ec"* eingeben.

4. Jedes Shell-Fenster hat einen Rahmen und eine Titelleiste. Man hat sich daran schon gewöhnt. Wie sieht es aber aus, wenn plötzlich keine Leiste und kein Rahmen mehr da ist. Wenn Sie es gerne wissen möchten, dann tippen Sie die folgende Zeile ein. Anschließend steht Ihnen der Befehl rahmenlos zu Verfügung.

```
Alias rahmenlos echo "*e[80u*e[0x*e[0y*e[31t"
```

   Nachdem Sie die Zeile eingegeben haben, ziehen Sie das Fenster auf seine maximale Größe auf. Geben Sie dann denn Befehl rahmenlos ein. Das Fenster hat sich kaum verändert. Erst durch gleichzeitiges Drücken der Ctrl-Taste und der l-Taste gefolgt von einem RETURN bringt das gewünschte Ergebnis. Der Rahmen ist weg. Falls es Ihnen so doch nicht gefällt, dann drücken Sie die Esc-Taste (im Shell-Fenster sehen Sie diese Eingabe nicht) und anschließend ein c. Durch Verkleinern des Fensters erhalten Sie wieder den Rahmen.

5. Es soll eine Datei auf dem Drucker ausgegeben werden. Gleichzeitig soll der Rechner weiter benutzbar bleiben. Die Datei soll also im Hintergrund gedruckt werden.

```
Alias Druck run copy to prt: [] clone
```

Die eckigen Klammern dienen als Platzhalter für den Parameter, der mit Druck eingegeben wird. Hier muß die Datei angegeben werden, die ausgedruckt werden soll, zum Beispiel

```
Druck sys:s/startup-sequence.
```

6. Es wird ein Befehl gesucht, der eine Datei "unsichtbar" macht, das heißt sie wird beim Dir- oder List-Befehl auf dem Bildschirm nicht aufgeführt.

```
Alias Hide protect [] +h
```

Mit der Eingabe des Befehls Hide und einem Dateinamen kann eine Datei versteckt werden. Dies geschiet mit dem Protect-Befehl. Mit ihm wird das H-Status-Bit der Datei gesetzt. Für die Benutzung dieses Befehls muß Ihr Rechner allerdings Kickstart 1.3 benutzen. Mit Kickstart 1.2 wird dieses H-Bit nicht berücksichtigt.

7. Wie unter 6. kann auch das Status-Bit s gesetzt werden. Sie müssen nur die folgende Zeile eingeben:

```
Alias SBit protect [] +s
```

Anschließend können Sie mit *SBit Dateiname* eine Batch-Datei ausführbar machen, ohne daß Sie den Execute-Befehl benutzen müssen, wobei der Execute-Befehl im c-Verzeichnis der Workbench vorhanden sein muß.

8. In Ihrer Shell-Startup-Datei ist ein Alias-Befehl schon vorhanden. Es ist der xCopy-Befehl. Das gleiche Prinzip können wir auch zum Löschen benutzen.

```
Alias xDelete delete [] all
```

Nach Eingabe dieser Zeile können Sie ein Verzeichnis samt dessen Inhalt löschen. Geben Sie dazu xDelete und ein Verzeichnis in die Shell ein.

9. Im Kapitel 6.3.2 haben wir eine Batch-Datei gezeigt, die einige CLI-Befehle in die RAM-Disk kopiert und diese dort unter einem verkürzten Befehlswort bereitstellt. Mit dem Alias-Befehl lassen sich die CLI-/Shell-Befehle wesentlich bequemer verkürzen.

```
Alias c copy
Alias p path
Alias d dir
Alias ex execute
Alias d delete
Alias t type
Alias r rename
Alias e echo
      .
      .
      .
```

Die einzelnen Befehle können nach Eingabe der Zeilen mit den entsprechenden Abkürzungen benutzt werden.

7. AmigaDOS und Multitasking

Was uns am Amiga immer wieder fasziniert hat, ist die Leistungsfähigkeit der Hard- und Software. Während andere Computerhersteller den Blitter (den Spezialbaustein für superschnelle Speicheroperationen) erst nach langen Vorankündigungen ausliefern, ist das System Amiga von vornherein damit ausgerüstet. Während andere Systeme bis zu 64 Farben zur Verfügung stellen, bietet der Amiga 4096 Farben gleichzeitig.

Aber noch viel leistungsfähiger als die Hardware des Amiga ist die mitgelieferte Software. Auch andere Computer arbeiten mit Fenstern, aber bei ihnen ist spätestens beim siebten Fenster Schluß, oder das Betriebssystem erlaubt zwar ein gewisses Multitasking (und davon ist in diesem Kapitel die Rede), aber eben nur ein äußerst eingeschränktes. Vor allem aber ist die mitgelieferte Software nicht in der Lage, Erweiterungen und Verbesserungen der Hardware voll auszunutzen.

Ganz anders steht hier die Software des Amiga da. Kommt demnächst der große, schnelle Bruder des 68000, der 68020, der Amiga ist schon darauf vorbereitet und das Betriebssystem unterstützt ihn. Werden Speicher-Erweiterungen auf mehrere MByte billiger? Der Amiga freut sich darauf. Bringt Commodore in Kürze eine Grafikkarte mit mehr Auflösung, Farben und Möglichkeiten? Die graphics.library des Amiga braucht kaum erweitert zu werden. Eine ganz besondere Stellung in dieser Leistungsfähigkeit des Amiga-Betriebssystems, die der Hardware weit voraus ist, nimmt das AmigaDOS ein. Warum - das werden Sie unter anderem in diesem Kapitel erfahren.

Auf einem Homecomputer echtes Multitasking ohne Einschränkung - AmigaDOS macht es möglich. Dabei ist dieser Teil des Betriebssystems auch noch so ausgelegt, daß er sich bei steigenden Möglichkeiten der Hardware um viele Funktionen erweitern läßt. Das konsequent verwirklichte Prinzip der Gerätetreiber ist ein wesentlicher Bestandteil dieser Flexibilität. Alle von DOS unterstützten Geräte, so unterschiedlich sie auch sein mögen,

werden mit den gleichen Routinen angesprochen. Die Umleitung von Ein- oder Ausgaben auf verschiedene Geräte ist eine wesentliche Voraussetzung für Multitasking. Dabei enthält das AmigaDOS als Untermenge die Befehle und Leistungen des weitverbreiteten MS-DOS der professionellen PCs und bietet zusätzlich weit mehr.

Ein kleiner Wermutstropfen steckt allerdings im AmigaDOS. Dieses Superbetriebssystem wird auf einem Rechner ausgeliefert, der noch lange nicht alle Möglichkeiten ausschöpft. Denn Multitasking auf einem Amiga 500 mit 512 KByte-Speicher und einem Laufwerk ist wie Porschefahren auf einem Acker, reizvoll, aber die vollen Möglichkeiten werden nicht ausgeschöpft. Auch der 68000-Prozessor ist ausreichend, aber AmigaDOS wäre auf einem 68020-Prozessor noch weit aus leistungsfähiger. Wir können nur hoffen, daß diese Möglichkeiten bald zum Standard jedes Amiga-Besitzers gehören.

7.1 Was ist Multitasking?

Vielleicht werden einige unter Ihnen bei dieser Fragestellung müde lächeln und sagen: "Das ist doch ein alter Hut. Von Multitasking spricht man, wenn der Rechner mehrere Sachen gleichzeitig macht". Aber so dumm ist die Frage gar nicht. Denn auch viele von denen, die Bescheid zu wissen meinen, arbeiten nur mit einem CLI, warten mit dem Weiterarbeiten, bis die Diskette fertig formatiert ist, oder starten erst dann erneut den Editor, wenn der C-Compiler mit dem Übersetzen fertig ist. Leider nutzen Sie dann die vollen Möglichkeiten des AmigaDOS in keiner Weise aus.

Multitasking ist etwas völlig Natürliches, was wir Menschen praktisch ständig machen und bei dem der Computer erst langsam nachzieht. Erstaunlicherweise benutzen viele trotzdem die Möglichkeiten des Rechners nicht, mehrere Dinge gleichzeitig zu tun, ohne auf das Ende der einen Tätigkeit zu warten. Nehmen wir als Vergleich eine typisch menschliche Aufgabe, an der sich auch gleich einige Probleme des Multitasking aufzeigen lassen:

Sie wollen ein leckeres Mittagessen kochen. Dieses soll aus einem Braten mit Pilzen und Zwiebeln, Kartoffeln und Spargel bestehen. Um 12 Uhr soll es auf dem Tisch stehen. Nun kämen Sie sicherlich nicht auf den Gedanken, alle Aufgaben nacheinander durchzuführen. Dann müßten Sie um 8 Uhr morgens die Kartoffeln aufsetzen, um dann gegen 11 Uhr 30 die Soße fertig zu machen und den Tisch zu decken. Natürlich wären Kartoffeln und Spargel bis dahin kalt. Statt dessen werden Sie sich überlegen, welche Aufgabe am längsten dauert und dementsprechend Ihre Zeit planen - und die anderen, kürzeren Aufgaben nebenbei erledigen. Sie setzen also um 11 Uhr den Braten auf den Herd, weil Sie wissen, daß er etwa eine Stunde benötigt. Anschließend setzen Sie das Wasser für die Kartoffeln auf, und während es heiß wird, schälen Sie die Kartoffeln. Wenn diese im kochenden Wasser sind, haben Sie 20 Minuten Zeit, um den Spargel fertigzumachen.

Bevor wir Ihnen jetzt das Wasser im Munde zusammenlaufen lassen, wollen wir lieber wieder den Bogen zum Multitasking machen. Trotzdem war das ein sehr typisches Beispiel für unser tägliches Multitasking, und man könnte unzählige weitere hinzufügen - vom Zigarettenanzünden beim Autofahren bis zum Zeitunglesen beim Fernsehen. Allen diesen Beispielen ist aber auch dieselbe Problematik gemeinsam: Beim Multitasking kann man sich schnell die Finger am kochenden Kartoffelwasser verbrennen, während der Braten anbrennt, oder das Heck des Vordermanns mit dem eigenen Auto demolieren, weil einem die Zigarette runterfällt. Genauso ist es auch beim Computer. Sie können relativ schlecht einen Text auf eine Diskette speichern, die gerade formatiert wird, und ebenso schlecht einen Text in eine Datei schreiben, die gerade auf den Drucker ausgegeben wird. Hier liegen Probleme, die man vermeiden sollte, und auf die wir in den folgenden Kapiteln genauer eingehen werden.

Multitasking ist also die Fähigkeit eines Computersystems, mehrere Aufgaben scheinbar gleichzeitig zu erledigen, oder sagen wir besser: mehrere Aufgaben in kurzen Zeitabschnitten hintereinander zu erledigen, so daß keine Aufgabe auf die andere warten muß und alle praktisch gleichzeitig fertig werden. Je weniger Probleme dabei entstehen, um so besser für uns Anwender

(Denken Sie nur an den Wagen des Vordermannes nach dem Zigarettenanzünden). Nun, glücklicherweise ist das AmigaDOS so ausgezeichnet programmiert, daß Zusammenstöße und verbrannte Finger beim Multitasking praktisch nicht vorkommen. So versucht AmigaDOS nicht, gleichzeitig zwei verschiedene Texte zu drucken, was der Lesbarkeit auch sehr widersprechen würde. Auch weigert sich AmigaDOS beharrlich, ein C-Programm auf Diskette zu ändern, das gerade von einem C-Compiler übersetzt wird. All dies haben wir der Firma Metacomco zu verdanken, die all diese Probleme vor uns erkannt und durch geschickte Programmierung abgefangen hat. Vielleicht haben Sie nun Spaß daran gefunden, mit uns zu überlegen, welche Zwischenfälle beim Multitasking eintreten können. Dann schauen Sie doch einmal im Kapitel 5.6 vorbei, wo wir solche Fälle auflisten und auch die Lösungsmöglichkeiten dafür anbieten. In den nächsten Abschnitten soll uns aber vielmehr interessieren, wie wir mit Multitasking komfortabler arbeiten, viel Zeit sparen und unsere Probleme lösen können.

7.2 Multitasking mit CLI und Workbench

Vielleicht haben Sie von der Benutzeroberfläche Workbench aus schon begeistert Freunden und Bekannten gezeigt, wie der Amiga mehrere Sachen gleichzeitig machen kann. Besonders beliebt und sofort verfügbar sind natürlich die Demoprogramme, die man nacheinander starten kann. Wenn es dann noch niemand wahrhaben will und an einen Trick glaubt, kann man ja immer noch dasselbe Programm mehrmals starten. Nur leider bleibt die Frage nach dem Nutzen. Lines und Boxes gleichzeitig ist zwar eine nette Sache, aber das könnte auch ein einziges Programm: in zwei verschiedenen Fenstern unterschiedliche Grafiken ablaufen lassen. Und wenn Sie einen Amiga 500 mit 512 KByte haben, werden Sie wohl auch kaum in die Verlegenheit kommen, jemand BECKERtext, DATAMAT und AmigaBASIC gleichzeitig zu zeigen. Trotzdem gibt es eine Fülle von nützlichen Anwendungen für Multitasking, und zwar in der Zusammenarbeit von Workbench und CLI.

Bevor wir Ihnen die vielen Anwendungsmöglichkeiten für Multitasking mit Workbench und CLI zeigen, müssen wir Ihnen leider sagen, daß die Workbench eigentlich gar nicht richtig "Multitasking-fähig" ist. Was, Sie glauben uns nicht? Dann formatieren Sie doch einmal mit *Initialize* eine Diskette und starten während des Formatierens ein Programm. Das geht nämlich nicht. Die ganze Zeit verwandelt sich der Mauspfeil in ein kleines Wölkchen (Bzzzzzzz), und läßt kaum weitere Aktionen zu. Also: Mehrere Programme nacheinander starten kann die Workbench, aber mehrere Aufgaben gleichzeitig durchführen leider nicht. Um so wichtiger ist es, daß es noch das CLI gibt, mit dem man wirklich mehrere Aufgaben gleichzeitig erledigen kann.

Wichtig ist natürlich, daß Sie auch als Besitzer eines Amiga 500 mit einem Laufwerk die wichtigsten CLI-Befehle im RAM haben. Was passiert, wenn Intuition versucht, ein Programm von der Diskette zu laden und AmigaDOS gleichzeitig nach dem gewünschten Befehl auf dieser Diskette sucht, ist nur mit Geduld zu ertragen. Sollten Sie dagegen stolzer Besitzer von zwei Laufwerken sein, so sollten Sie in einem Laufwerk die Systemdiskette mit den CLI-Befehlen haben und mit der Workbench von der anderen Diskette aus das Programm starten. Wie CLI-Befehle ins RAM kopiert werden und wie man sie dann verfügbar macht steht - falls Sie es vergessen haben sollten, in Kapitel 4 - unter Batch-Dateien. Nun wollen wir aber endlich zu den Anwendungsmöglichkeiten und der Zusammenarbeit mit der CLI-Workbench kommen.

Sollten Sie noch immer eine Kopie der Originaldisketten mit deren Startup-Sequence verwenden, so bitten wir Sie inständig, diese zu ändern. Die vorgesehene Startup-Sequence beendet das CLI am Schluß mit

```
ENDCLI >NIL:
```

was das einzige vorhandene CLI im wahrsten Sinne des Wortes sang- und klanglos verschwinden läßt: Selbst die Abschlußmeldung wird ins "Nichts" geschickt. Legen Sie also eine Kopie der Workbench-Diskette ins Laufwerk df0:, laden Sie die Startup-Sequence mit

```
ED DF0:s/Startup-Sequence
```

und löschen Sie diese Zeile. Anschließend können Sie die geänderte Datei mit ESC und X abspeichern und behalten von nun an Ihr wichtigstes Hilfsmittel bei der Arbeit mit dem Amiga. Was können Sie sinnvollerweise mit Workbench und CLI gleichzeitig anfangen?

Nun, vor allen anderen Möglichkeiten und Tips ist ein stets vorhandenes CLI eine Art Hintertür für echte Probleme. Stellen Sie sich einmal vor, Sie haben einige wichtige Dateien in der RAM-Disk und starten ein Programm von der Workbench aus. Dies Programm hängt sich während des Ladens auf, was durchaus häufig passiert, vor allem, wenn das Programm beispielsweise mit der Speicher-Erweiterung auf 1 MByte nicht klarkommt. Dann können Sie von der Workbench aus nichts mehr machen. Diese zeigt nur noch das "Bzzzzzzzz" und läßt damit auch ein mögliches Laden des CLI nicht mehr zu. Sie können Ihren Amiga nur noch durch ein Reset aus der Zwangslage befreien, aber die Dateien in der RAM-Disk sind natürlich verloren. Hätten Sie jetzt aber ein CLI-Fenster hinter allen anderen Fenstern versteckt, so könnten Sie es einfach nach vorne klicken und die wichtigen Dateien aus der RAM-Disk mit

```
COPY RAM: DF0: ALL
```

auf die Diskette retten. Natürlich hilft dieses rettende CLI nicht gegen die roten blinkenden Balken, mit denen sich die Guru Meditation meldet. Aber selbst bei einem harmloseren Requester mit der Meldung

```
Task held - finish all disk aktivities ......
```

sind Sie ohne ein rettendes CLI-Fenster meist ziemlich aufgeschmissen. Also ein klitzekleines CLI-Fenster im Hintergrund kostet wirklich nicht viel Speicher und erspart viel Ärger und Enttäuschung.

Eine weitere wichtige Anwendung für die Zusammenarbeit Workbench-CLI ist das Kopieren von mehreren Dateien. Sie arbeiten gerade mit BECKERtext, ein Freund kommt vorbei und bittet Sie, mal eben einige Ihrer Texte auf seine Diskette zu ko-

pieren. Sie könnten nun natürlich das BECKERtext-Fenster ganz klitzeklein machen und anschließend von der Workbench aus alle Dateien auf seine Diskette "schieben". Einfacher ist es aber, daß CLI-Fenster nach vorne zu klicken, mit Copy alle Dateien auf seine Diskette zu kopieren und währenddessen noch eben den Absatz zu Ende zu schreiben. Und sollte seine Diskette noch nicht formatiert sein, sparen Sie noch mehr Zeit und Unterbrechungen: Sie geben beispielsweise

```
FORMAT DRIVE DF0: NAME TEXTE
```

ein, und wenn der Cursor in der nächsten Zeile steht, können Sie schon den nächsten Befehl eingeben:

```
COPY DF1:TEXTE/#? DF0:
```

Sie müssen bei der weiteren Arbeit nicht darauf warten, daß der Amiga mit dem Formatieren fertig ist. Sofort beginnt er, die eben formatierte Diskette mit den Texten zu füllen, während Sie natürlich Ihrem Freund das neuerworbene Spiel zeigen können.

Das eben Gesagte gilt natürlich in ähnlicher Weise für das Löschen von mehreren Dateien. Sie haben in einem gesonderten Verzeichnis Sicherheitskopien jeweils ein Backup von jedem geschriebenen Text erstellt. Nun wird langsam der Platz auf Ihren Disketten eng und Sie brauchen die Sicherheitskopien nicht mehr. Wenn Sie von der Workbench aus die gewünschte Schublade öffnen, alle Texte markieren und mit Disgard löschen, nimmt allein das Anzeigen der Dateien eine ganze Zeit in Anspruch - vom Markieren und der Zeit für das Löschen einmal abgesehen. In dieser Zeit können Sie mit der Workbench nicht mehr arbeiten. Viel einfacher und schneller ist es da, das CLI-Fenster nach vorne zu klicken,

```
DELETE DF0:Sicherheitskopien/#?
```

alle Dateien in dem Verzeichnis zu löschen und währenddessen weiter mit der Workbench arbeiten zu können.

Vielleicht stört Sie bei der Zusammenarbeit Workbench-CLI die Lage und Größe des voreingestellten CLI-Fensters. Dieses muß erst verkleinert und beiseite geschoben werden, bevor Sie mit

der Workbench sinnvoll arbeiten können. Leider können Sie die Größe dieses voreingestellten Fensters nicht so einfach ändern, ohne erst zur Maus zu greifen. Trotzdem gibt es eine einfache Möglichkeit, das gewünschte CLI-Fenster zu erhalten. Öffnen Sie einfach ein neues Fenster in den gewünschten Ausmaßen, und schließen Sie anschließend das alte CLI. Dieser Teil der Startup-Sequence könnte dann folgendermaßen aussehen:

```
.....
........
NEWCLI "CON:10/10/10/10/MeinCLI"
ENDCLI
........
.....
```

7.3 Multitasking mit NewCLI

Sie haben sich inzwischen hoffentlich daran gewöhnt, nicht nur stets ein CLI-Fenster zur Verfügung zu haben, sondern führen auch wichtige Aufgaben quasi nebenbei mit dem CLI durch. Aber auch dann können Sie noch einiges an Zeit und Effizienz gewinnen. Nehmen wir einmal folgenden einfachen Fall, der gar nichts mit Multitasking zu tun hat: Sie haben heute 5 neue Texte geschrieben und im Verzeichnis df0:Texte abgelegt. Dort gibt es inzwischen 40 Texte, und die 5 neuen Texte wollen Sie noch eben per CLI auf den Drucker ausgeben. Vorher wollen Sie diese aber noch einmal kurz anschauen, ohne erst die Textverarbeitung laden zu müssen. Sie wissen natürlich, daß das ganz einfach mit der Anweisung:

```
TYPE df0:textname
```

geht, und das anschließende Drucken des Textes ist ganz einfach mit

```
COPY DF0:textname PRT:
```

zu bewerkstelligen. Doch da taucht schon das erste Problem auf: Sie haben die Namen der Texte nicht mehr so genau im Kopf. Also schauen Sie mit *CD df0:Texte* in das Verzeichnis zum aktuellen Directory und sehen sich mit *Dir* den ersten Textnamen

an. Aha! Der Name des Textes war "Peter1.10.87". Nun können Sie sich den Text noch einmal kurz mit

```
TYPE Peter1.10.87
```

anschauen und schließlich auf den Drucker ausgeben. Der ist schnell fertig, aber hoppla - wo sind denn die anderen Textnamen geblieben? Durch das *Type* sind die File-Namen natürlich längst nach oben verschwunden und Sie müssen erneut *Dir* eingeben, um den nächsten Namen zu erhalten. Jetzt können Sie natürlich - aus Erfahrung klug geworden - mit *Dir* die Namen auf den Bildschirm holen und fein säuberlich abschreiben. Anschließend können Sie dann Datei für Datei anschauen und ausdrucken. Aber da gibt es eine wesentlich einfachere Möglichkeit.

Starten Sie mit NewCLI ein neues CLI. Verschieben Sie das zweite Fenster so auf dem Bildschirm, daß es etwa die obere Hälfte einnimmt. Geben Sie dort anschließend *Dir* ein, so daß die Textnamen im Fenster sichtbar sind (Gegebenenfalls können Sie die Bildschirmausgabe durch Drücken einer beliebigen Taste anhalten). Nun können Sie im ersten CLI die Textnamen in Ruhe lesen, die Texte mit Type auf dem Bildschirm anzeigen lassen und auf den Drucker ausgeben. Trotzdem verschwinden die Namen der folgenden Texte nicht einfach am oberen Bildschirmrand. Das Problem ist komfortabel gelöst. Ein zweites CLI zu starten, und dort die Bildschirmausgabe vorzunehmen, das ist immer dann eine praktische Sache, wenn man mit dieser Information weiterarbeiten will. Das gilt genauso für das Löschen oder Kopieren von mehreren Texten. Lassen Sie sich diese in einem Fenster (CLI) anzeigen, und führen Sie im zweiten Fenster die Aktionen durch. Wenn Sie das zweite Fenster dann nicht mehr benötigen, können Sie es ja problemlos durch EndCLI verschwinden lassen.

Nun wollen wir aber die Annehmlichkeit noch etwas weiter treiben. Wir gehen einmal davon aus, daß die Texte ziemlich lang sind und jeder Druck einige Minuten dauert. Dann müssen Sie in unserem Beispiel diese Zeit abwarten, bevor Sie mit den weiteren Textdateien fortfahren können. Machen Sie doch in so einem Fall einfach ein weiteres CLI auf. Im ersten Fenster können Sie dann die Textnamen lesen, im zweiten Fenster mit Type die

Texte kontrollieren und währenddessen im dritten Fenster schon einen Text drucken lassen. Eins wird Ihnen dabei sicherlich verlorengehen: die gemütliche Kaffeepause, die Sie immer einlegen konnten, während der Computer mit Arbeit beschäftigt war. Diese Kaffeepausen gibt es bei AmigaDOS nicht mehr zwangsweise, sondern nur noch auf Ihren ausdrücklichen Wunsch hin.

Wir hoffen, daß Sie nun schon einen kleinen Vorgeschmack auf die vielfältigen Möglichkeiten von AmigaDOS und Multitasking bekommen haben. Aber es soll natürlich nicht bei einer kleinen Vorspeise bleiben. Vielmehr wollen wir Ihnen zeigen, wie Sie insgesamt in wesentlich kürzerer Zeit Ihre Aufgaben erledigen können. Wenn Sie häufig mit dem CLI arbeiten und nur wenig Speicher übrig haben, sollten Sie eigentlich immer mit einem zweiten CLI arbeiten. Hier gilt dasselbe wie bei der Workbench: Ist das übliche CLI einmal länger beschäftigt oder hat es sich gar durch ein Programm aufgehängt, so können Sie praktisch problemlos mit dem zweiten CLI-Fenster weiterarbeiten. Uns ist es zu Anfang oft passiert, daß wir nur mit einem CLI gearbeitet haben und dann "versehentlich" *Copy text PRT:* eingegeben haben. Dann mußten wir warten oder erst umständlich über die Workbench und verschiedene Fenster ein weiteres CLI-Fenster öffnen. Wie gut, wenn Sie dann ein zweites CLI haben und einfach weitermachen können.

Bei der Arbeit mit neuen CLI-Windows sollten Sie von vornherein die Möglichkeiten des AmigaDOS berücksichtigen. Angenommen, Sie haben als aktuelles Verzeichnis df0:Texte/Privat gewählt und öffnen ein neues CLI mit NewCLI. Dann werden Eingabe und Ausgabe automatisch auf die Devices oder Directorys des vorherigen CLI gesetzt. In unserem Fall hat also das 2. CLI als Current Directory *(CD)* ebenfalls *df0:Texte/Privat*, und Sie müssen das Verzeichnis nicht neu angeben. AmigaDOS denkt eben mit.

Genauso komfortabel ist die Möglichkeit, die Außmaße und Lage des neuen CLI-Fensters gleich mit anzugeben. So können Sie beispielsweise mit

```
NEWCLI "CON:10/10/10/10/MeinCLI"
```

ein klitzekleines CLI-Fenster oben links erzeugen. Das kann Sie nun wirklich nicht bei der Arbeit stören. Am besten tippen Sie diese lange Zeile nur einmal mit dem ED ab und legen sie unter dem Namen NC (für NewCLI) auf der Diskette ab. Dann können Sie dieses neue CLI-Fenster einfach mit

```
EXECUTE df0:NC
```

aufrufen. Haben Sie dann auch noch den Execute-Befehl in E umbenannt und ist df0: das aktuelle Verzeichnis, so erscheint dieses zweite CLI sogar schon durch die Zeile:

```
E NC
```

7.4 Multitasking mit Run

Die Möglichkeit, mit NewCLI so viele verschiedene Aufgaben wie nötig abarbeiten zu lassen, ist sicherlich interessant und oft eine große Hilfe und Zeitersparnis. Aber spätestens beim vierten CLI-Fenster tauchen zwei Probleme auf:

- Obwohl der Amiga mehrere "Bildschirme" gleichzeitig verwalten kann, bleibt natürlich letztlich die Beschränkung auf die Größe des Monitors, weil alle CLI-Befehle mit Fenstern auf dem Workbench-Screen arbeiten. Dadurch nehmen sie natürlich schnell die ganze Fläche ein und überdecken sich noch gegenseitig. Und besonders angenehm ist es auch nicht, ständig verschiedene Fenster nach vorne oder hinten klicken zu müssen.

- Besonders auf einem Amiga 500 mit 512 KByte Speicher kostet jedes Fenster eine ganze Menge des kostbaren freien Speichers, und dann bleibt bei mehreren CLI-Fenstern bald kaum noch etwas übrig.

Um mehrere Aufgaben gleichzeitig erledigen zu können, ohne für jede Aufgabe auch ein Fenster zu benötigen, wurde der Befehl Run entwickelt. Mit diesem Befehl richten Sie ein neues CLI ein, was allerdings völlig ohne ein eigenes Fenster auskommt. Probieren Sie das doch am besten einmal aus. Sie haben

die Workbench-Diskette mit den CLI-Befehlen in DF0? Dann geben Sie folgende zwei Zeilen ein:

```
RUN COPY DF0:C/RUN RAM:C
DIR DF0:
```

Haben Sie die erste Zeile mit Return abgeschlossen, erscheint eine Zeile

```
[CLI 2]
```

und anschließend wieder die "1>" (Wenn Sie natürlich schon mehrere CLI-Prozesse gestartet haben, erscheinen bei Ihnen andere Zahlen). Anschließend haben Sie (hoffentlich schnell genug - während die Floppy noch arbeitet) die zweite Zeile eingegeben und das CLI beginnt, das Inhaltsverzeichnis anzuzeigen. Dieses Beispiel macht noch nicht allzuviel Sinn, zeigt aber schon die grundsätzliche Anwendungsmöglichkeit für den Befehl Run: Alle Aufgaben, die eventuell länger dauern und nichts auf dem Bildschirm ausgeben, lassen sich gut mit Run aufrufen.

Während die erste Einschränkung noch relativ einleuchtend ist - dauert die Abarbeitung nur eine Sekunde, so können Sie kaum schnell genug die nächste Zeile tippen -, so leuchtet die zweite Einschränkung nicht sofort ein. Deshalb ein kleines Beispiel für zwei gleichzeitig ablaufende Aufgaben, von denen die eine mit Run gestartet worden ist. Geben Sie direkt hintereinander folgende Zeilen ein:

```
TYPE DF0:S/Startup-Sequence
Dir DF0:
```

Im ersten Augenblick scheint alles ganz normal zu verlaufen, aber plötzlich erscheinen in der Liste Ihrer Startup-Befehle Teile des Directorys von df0. Das liegt daran, daß ein mit Run gestarteter CLI-Befehl zur Ausgabe ja kein Fenster zur Verfügung hat. Daher gibt er alle Ausgaben auf dem CLI-Fenster aus.

Trotz dieser Einschränkungen gibt es vielfältige Anwendungen für die Nutzung des Multitasking mit Run. Dazu gehören natürlich erst einmal alle CLI-Befehle, die normalerweise keine Bild-

schirmausgabe machen und relativ viel Zeit benötigen (Format, DiskCopy, Copy). So kann man beispielsweise mit

```
RUN COPY Text PRT:
```

problemlos einen Text ausdrucken und sofort weiterarbeiten. Allerdings sollten Sie insbesondere als Besitzer eines Amiga mit einem Laufwerk daran denken, daß Sie während des Druckens nicht einfach die Diskette mit dem Text aus dem Laufwerk nehmen können. Wollen Sie auch während des Druckens mit anderen Disketten weiterarbeiten, so sollten Sie den Text erst in die RAM-Disk kopieren und von dort aus drucken lassen.

Eine ganz wichtige Aufgabe hat der Befehl Run, wenn Sie vom CLI aus ein Programm starten wollen und mit diesem längere Zeit arbeiten möchten. Natürlich könnten Sie mit NewCLI ein neues CLI starten und von dem zweiten CLI das Programm dann mit

```
Programm
```

aufrufen. Dann haben Sie aber während der ganzen Arbeitszeit ein völlig nutzloses Fenster, das nicht nur Platz, sondern auch Speicher kostet. Einfacher ist es deshalb, vom ersten CLI aus das gewünschte Programm mit

```
RUN Programm
```

zu starten. Ein typisches Beispiel stellt die Benutzung von Editoren dar. Sie schreiben beispielsweise ein C-Programm und wollen immer wieder Erweiterungen des Programms austesten. Haben Sie nur ein CLI und starten das Programm direkt, so müssen Sie für jeden Testdurchlauf des Compilers erst den Editor verlassen, den Compiler aufrufen, um dann anschließend wieder den Editor zu starten. Auch ein Starten des Editors von einem neuen CLI aus ist problematisch wegen des verbrauchten Platzes und Speichers. Rufen Sie dann den Editor einfach mit

```
RUN ED Programm.c
```

auf. Nach dem Abspeichern mit *ESC + sa* können Sie im weiterhin benutzbaren CLI mit *Run* den Compiler aufrufen, und bei

dessen erster Fehlermeldung gleich im Editor mit der Kontrolle der Zeile beginnen. So sparen Sie viel Zeit.

Allerdings gibt es bei Run noch eine weitere Einschränkung, die im Einzelfall unangenehme Folgen haben kann. Einem mit Run gestarteten Programm fehlen sowohl eigene Ausgabe- als auch Eingabefenster. Die Ausgabe kann noch ins bisherige Fenster erfolgen, aber woher könnte eine Eingabe kommen? Sicherlich haben Sie schon gemerkt, daß die Eingabe streng an Fenster gebunden ist. Jedes Programm, das eine Eingabe über die Tastatur bekommen will, benötigt ein Fenster. Daher sind Eingaben an zwei unabhängige Programme durch ein Fenster nicht möglich. Wie sollte auch der Amiga wissen, an welches Programm er jetzt die Eingabe weitergeben soll?

Sie werden jetzt vielleicht sagen: "Aber die CLI-Befehle benötigen doch gar keine Eingaben mehr". Das stimmt nicht so ganz. Denn eine Eingabe lassen die CLI-Befehle meist zu: den Abbruch durch Ctrl und C oder andere Ctrl-Funktionen. Wenn Sie aber ein Programm oder einen CLI-Befehl mit Run starten, geht jedes Ctrl und C nicht etwa an den so gestarteten Befehl, sondern an das bisherige CLI, das damit wenig anzufangen weiß. Stellen Sie sich nun einmal vor, Sie haben einen 30 KByte langen Text und wollen sich den Anfang kurz einmal mit

```
RUN TYPE Text
```

ausgeben lassen. Das Anhalten der Ausgabe funktioniert auch - Drücken einer Taste -, aber der Abbruch mit Ctrl und C zeigt keine Wirkung mehr. Dieses Problem haben natürlich auch die Entwickler des AmigaDOS gesehen und eine entsprechende Lösung eingebaut. Diese steckt im Befehl *Break*. Mit ihm können Sie auch an mit Run gestartete Befehle ein Ctrl und C schicken. Starten Sie also die Textausgabe vom ersten CLI-Fenster aus mit:

```
RUN TYPE Text
```

so können Sie jederzeit

```
BREAK 2
```

eingeben und damit die Textausgabe durch das zweite CLI beenden. Die genaueren Möglichkeiten entnehmen Sie bitte der Beschreibung in Kapitel 3, uns kommt es hier nur darauf an, die grundsätzliche Abbruchmöglichkeit für mit *Run* gestartete Befehle zu zeigen.

Zu Schluß dieses Abschnittes wollen wir den Interessierten noch einige grundsätzliche Informationen zum Unterschied zwischen NewCLI und Run geben. Haben Sie unseren neuen CLI-Befehl *Task* aus Kapitel 7 schon abgetippt? Er zeigt im Gegensatz zum Status-Befehl nicht nur die vorhandenen CLI-Prozesse, sondern alle im Amiga vorhandenen Tasks an. Jedes NewCLI erzeugt einen neue Task mit dem Namen New CLI, während jedes Starten mit Run eine Task namens Background CLI erzeugt. Das wollen wir Ihnen bei einer Ausgabe aller Tasks kurz zeigen. Diese könnte beispielsweise so aussehen:

```
CON   5
Initial CLI   0
input.device   20
File System   10
Workbench   1
trackdisk.device   5
CON   5
Background CLI   0
```

In diesem Beispiel erscheint in der letzten Zeile ein Eintrag Background CLI. Hier läuft also gerade ein mit Run gestarteter Befehl, der kein eigenes Ein- und Ausgabefenster hat.

```
CON   5
New CLI   0
CON   5
Initial CLI   0
Workbench   1
File System   10
input.device   20
trackdisk.device   5
RAM   0
CON   5
```

In diesem Beispiel steht an zweiter Stelle ein New CLI. Hier war ein zweites CLI-Fenster geöffnet worden. Dadurch kommen Tastatureingaben aus diesem Fenster.

7.5 Wie man das CLI am besten einsetzt

Wahrscheinlich haben Sie schon in den Kapiteln 5.2 bis 5.4 einen Eindruck davon bekommen, wieviel Zeit und Arbeit Sie durch die geschickte Nutzung der vollen Möglichkeiten des AmigaDOS sparen können. Besonders wichtig ist dabei, daß Sie über die unterschiedlichen Möglichkeiten und deren Wirkung Bescheid wissen und diese optimal einsetzen können. Das beginnt schon bei der Arbeit mit der Workbench. Zwar können Sie hier viele Aufgaben komfortabel lösen, aber durch den geschickten Einsatz des CLI können Sie in vielen Fällen längere Wartepausen beim Formatieren und Kopieren vermeiden. Zusätzlich können Sie natürlich durch ein CLI neben der Workbench viele Dinge möglich machen, die von der komfortablen Oberfläche aus nicht möglich sind.

So können Sie mit dem CLI nicht nur Dateien sehen, die kein .info-File haben, sondern darüber hinaus auch in Unterverzeichnissen nachsehen, die ebenfalls kein Icon für eine Schublade besitzen. Das ist z.B. dann wichtig, wenn Sie von einem Freund eine Diskette mit interessanten Programmen bekommen, von der Workbench aus aber nichts auf der Diskette sehen können.

Arbeiten Sie mehr mit dem CLI, so sollten Sie den richtigen Einsatz der CLI-Prozesse planen. Bei der Benutzung von Befehlen, die Ausgaben in einem Fenster produzieren, sollten Sie den Run-Befehl nicht benutzen. Überhaupt ist es äußerst nützlich, stets ein zweites CLI-Fenster zur Sicherheit im Hintergrund zu haben. Es passiert schnell, daß man mal eben einen Text auf den Drucker ausgeben oder eine Reihe von Dateien kopieren will, und hat man nur ein CLI-Fenster, kann man sich nur zwischen Warten und Abbruch entscheiden. Da kann man besser ein vorhandenes zweites CLI nach vorne klicken und mit diesem weiterarbeiten. Diese Notwendigkeit tritt viel häufiger ein, als man gemeinhin annimmt.

Eine besondere Bedeutung beim konsequenten Einsatz des Multitasking mit dem CLI nimmt die richtige Planung der vorhandenen Geräte ein. Das nimmt besonders drastische Formen an, wenn Sie nur im Besitz eines Diskettenlaufwerkes sind und nur

über einen 512 KByte Speicher verfügen. Dann müssen Sie natürlich mit einigen Einschränkungen leben. Es bringt nicht allzuviel, wenn Sie zwar drei CLI-Fenster mit drei verschiedenen Aufgaben betreut haben, der Amiga Sie aber für die nächste Zeit nur noch mit Diskettenwechsel beschäftigt. Hier sollten Sie möglichst schnell feststellen, welche CLI-Befehle Sie am häufigsten benötigen und diese bald in die RAM-Disk kopieren. Dadurch können Sie dann beispielsweise den zu druckenden Text im Laufwerk df0 haben und drucken, während die anderen CLI-Prozesse ihre Befehle aus der Main-Disk holen.

Natürlich ist es auf diese Weise nicht möglich, mit einem Laufwerk Texte von zwei verschiedenen Disketten zu drucken. Hier müssen Sie von vornherein planen, also beispielsweise die Texte schon auf einer Diskette gespeichert haben oder auch dafür, die RAM-Disk als ständig verfügbares Speichermedium einsetzen.

Besitzer der 1-MByte-Erweiterung haben es da schon wesentlich einfacher. Sie können nicht nur freizügiger Befehle und notwendige Programme im RAM bereithalten, sondern auch zusätzlich noch notwendige Dateien ins RAM kopieren. Ebenfalls günstig ist der Einsatz eines Amiga ohne Speicher-Erweiterung mit einem zusätzlichen Laufwerk. Dann bleibt die Workbench-Diskette in df0, und die Daten liegen in df1. Dadurch bleibt viel Speicherplatz für Programme erhalten.

Ebenso müssen Sie den Einsatz der verschiedenen Ausgabemedien geschickt planen. So wie es kaum sinnvoll ist, zwei Dateien gleichzeitig auf dem selben Bildschirm auszugeben, erzeugen zwei gleichzeitig auf einen Drucker ausgegebene Texte kaum lesbare Seiten. Während dies aber beim Bildschirm möglich ist (Einmal Type mit Run starten und ein zweites Type vom ursprünglichen CLI), verhindert das AmigaDOS die gleichzeitige Ausgaben mehrerer Prozesse oder Tasks auf eine Schnittstelle (Drucker, RS 232). Daher sollten Sie zuerst einen Text fertigschreiben, diesen vom CLI aus drucken und mit der Textverarbeitung dann den zweiten Text zu Ende schreiben.

Wollen Sie mehrere Ergebnisse ausdrucken, und dauert das Erstellen der Ergebnisse recht lange, so ist es meist günstig, die

Ergebnisse in Dateien schreiben zu lassen. So können gute Textverarbeitungen Texte meist in Dateien drucken. Wenn der Text mit allen Druckersteuercodes in eine Datei gedruckt ist, können Sie ihn vom CLI aus zum Drucker schicken, während die Textverarbeitung den nächsten Text in eine Datei druckt.

Besonders bei langen Verzeichnissen mit vielen Dateien dauert es oft recht lange, bis *Dir* sie angezeigt hat. Wenn Sie die Information nicht sofort benötigen, können Sie die Ausgabe problemlos in eine Datei leiten und diese Datei später schnell mit *Type* auf dem Bildschirm anzeigen lassen. Das geht immer noch viel schneller, als auf das Ergebnis zu warten. Geben Sie dazu einfach ein:

```
RUN DIR >ram:Inhalt DF0:C
```

und später können Sie sich mit

```
TYPE RAM:Inhalt
```

sehr schnell das Inhaltsverzeichnis des langen Verzeichnisses C: anzeigen lassen. Das *Run* ist natürlich sehr wichtig, weil wir sonst auf das Abarbeiten des Befehls *Dir* warten müßten, und das wollten wir ja gerade vermeiden.

Eine besonders effiziente Ausnutzung des Multitasking können Sie auch durch geschickten Einsatz der Batch-Dateien erzielen. Wir haben inzwischen eine recht komfortable, aber auch lange Startup-Sequence. Es dauert im Schnitt 2 Minuten, bis diese abgearbeitet ist. Warum sollten wir in dieser Zeit warten? Im einfachsten Fall fügen Sie einfach ganz zu Anfang ein

```
NEWCLI
```

in die Startup-Sequence ein. So haben Sie schon sehr schnell ein CLI-Fenster, mit dem Sie arbeiten können. Leider gibt es da noch gewisse Einschränkungen, weil beispielsweise die deutsche Tastaturanpassung nicht aufgerufen ist. Deshalb sollten Sie zumindest ein *df0:System/Setmap df0:Devs/Keymaps/d* einfügen. Natürlich können Sie auch etwas komfortabler vorgehen und für das neue CLI eine eigene kleine Startup-Sequence aufrufen, beispielsweise mit einer ersten Zeile namens:

```
EXECUTE DF0:S/MeineSequence
```

In der Datei könnten dann Befehle für den deutschen Zeichensatz und weitere Vorgaben stehen. In jedem Fall könnten Sie dann sofort mit einem losgelösten CLI zu arbeiten beginnen, bevor die komplette Startup-Sequence abgearbeitet ist.

7.6 Tiefer Eingriff mit ChangeTaskPri

In den letzten Kapiteln haben Sie verschiedene Möglichkeiten kennengelernt, Aufgaben gleichzeitig erledigen zu lassen. Dadurch kann viel Wartezeit eingespart werden. Trotzdem tauchen dabei leider noch einige Probleme auf, die die volle Nutzung dieser Möglichkeiten verhindern. Nehmen wir ein Beispiel:

Der Editor ED ist beim Bildschirmaufbau nicht gerade einer der schnellsten. Wenn Sie nun eine arbeitsintensive Aufgabe nebenbei erledigen lassen, werden Sie feststellen, daß die Arbeit mit dem Editor immer wieder kurz stockt. In diesen Phasen ist die andere Task an der Reihe und zeigt uns damit deutlich, daß kein Amiga mehrere Aufgaben gleichzeitig erledigen kann. Vielmehr wird alles in kleinen Zeitabständen hintereinander erledigt. Das kann für eine wichtige Aufgabe allerdings sehr störend sein. Daher kann das Amiga-Betriebssystem jeder Aufgabe eine bestimmte Wichtigkeit zuordnen. Aufgaben mit hoher Wichtigkeit werden bevorzugt behandelt, Aufgaben mit kleinerer Wichtigkeit hinterher.

Angenommen, Sie erstellen einen Text mit dem ED und wollen ihn zwischendurch schon einmal ausdrucken, um einen ungefähren Überblick über das Geschriebene zu bekommen. Dann ist es Ihnen doch sicherlich nicht allzu wichtig, ob dieser Text nach ein oder zwei Minuten ausgedruckt ist. Vielmehr wollen Sie beim Weiterschreiben nicht darauf warten müssen, bis der Editor endlich das nächste Zeichen auf den Bildschirm ausgibt. Zum Vergeben solcher Prioritäten wurden unterschiedliche Prioritäten eingeführt. Für CLI-Prozesse gibt es den Befehl ChangeTaskPri, um ihre Priorität zu verändern. Dieser Befehl setzt die Priorität der CLI-Tasks, von dem aus er aufgerufen wird, auf einen Wert

zwischen -128 und +127. Probieren Sie das doch einfach einmal aus, indem Sie sich zuerst mit

```
STATUS FULL
```

die aktuellen Informationen über die CLI-Prozesse anzeigen lassen. Bei einem CLI sieht diese Anzeige etwa folgendermaßen aus:

```
Task  1: stk 1600, gv 150, pri 0 Loaded as command: status
```

Die für uns wichtige Angabe ist das pri 0. Dies ist die Priorität unseres Prozesses und gleichzeitig der Standardwert, mit dem alle CLIs automatisch starten. Nun ändern wir diesen Wert einmal auf 3 ab:

```
CHANGETASKPRI 3
```

Wenn wir uns anschließend die Informationen der Prozesse anzeigen lassen, erhalten wir folgende Anzeige:

```
Task  1: stk 1600, gv 150, pri 3 Loaded as command: status
```

An dem Eintrag pri 3 sehen wir, daß sich etwas geändert hat. Unser CLI hat seine eigene Wichtigkeit um 3 erhöht. Nun erwarten Sie vielleicht, daß die ganze Welt stehenbleibt oder zumindest der gesamte Amiga, um auf unseren CLI zu warten. Das ist natürlich nicht so und darf auch nicht sein. Denn der Amiga hat ein Multitasking-Betriebssystem, und da dürfen im Normalfall nicht alle Aufgaben auf das Erledigen einer warten. Nun kommt eine Task mit der höheren Priorität eher an die Reihe als eine Task mit einer niedrigeren Priorität. Das können wir auch schnell an einem Beispiel ausprobieren. Erzeugen Sie mit NewCLI ein zweites CLI, und verschieben Sie die Fenster der beiden CLI so, daß das erste Fenster die untere Hälfte einnimmt, das zweite die obere Hälfte des Bildschirms. Nun wollen wir einmal in beiden CLI-Fenstern den Unterschied in der Abarbeitung beobachten, wenn verschiedene Prioritäten vorgegeben worden sind.

Geben Sie also im unteren Fenster *ChangeTaskPri -127* und im oberen CLI-Fenster *ChangeTaskPri 127* ein. Nun kommt es auf

das genaue Starten und Stoppen der Abläufe an. Schreiben Sie in beide CLI-Fenster die Befehlszeile (ohne Return zu drücken!):

```
LIST DF0:C
```

Nun müssen Sie möglichst schnell in beiden Prozessen die Befehle anordnen. Dazu klicken Sie mit der Maus im unteren Fenster, setzen die Maus schon mal ins obere Fenster, drücken Return, klicken oben und drücken wieder Return.

Puh, haben Sie das schnell genug geschafft? Dann können Sie jetzt in den Fenstern wunderschön verfolgen, wie mit unterschiedlicher Geschwindigkeit das Inhaltsverzeichnis von df0:C angezeigt wird. Trotzdem ist der Geschwindigkeitsunterschied noch nicht allzu berauschend. Vielleicht hätten Sie sich eine viel größere Differenz vorgestellt und gewünscht. Das ist aber bei der Art der Programmierung des Betriebssystems nicht möglich und auch nicht erwünscht. Sobald eine Task vom Betriebssystem eine Aktion fordert und darauf wartet, wird er in eine Art Schlafzustand (Waiting-Liste) versetzt und kostet keine Rechenzeit mehr. Dann haben auch Tasks mit einer viel tieferen Priorität die Möglichkeit, an die Reihe zu kommen. Das ist beispielsweise der Fall, wenn eine Task eine Spur der Diskette einlesen läßt. Er geht solange in die Warteliste, bis die Spur vollständig gelesen ist.

Trotz dieser scheinbar geringen Unterschiede, die wir nur der geschickten Programmierung des Amiga-Betriebssystems zu verdanken haben, können Sie den Unterschied in der Bearbeitung viel drastischer feststellen. Geben Sie im CLI 1 *ChangeTaskPri 0* ein, und erzeugen Sie einen zweiten Prozeß mit

```
RUN ED RAM:Unterschied
```

In dem leeren Fenster des Editors drücken Sie ca. 20mal Return, damit Sie den Cursor innerhalb des Windows frei bewegen können. Dann verschieben Sie das CLI-Fenster so, daß Sie dort neben dem Editor ebenfalls Eingaben machen können. Lassen Sie sich dann in diesem Fenster mit

```
DIR DF0:C
```

das umfangreiche Inhaltsverzeichnis anzeigen, und versuchen Sie gleichzeitig, den Cursor innerhalb des Editorfensters zu bewegen. Sie werden feststellen, daß das recht mühsam ist und er immer wieder stockt. Außerdem können Sie ihn kaum gezielt an eine Position bewegen, weil er erst nicht vorwärtskommt und dann plötzlich wieder weit über das Ziel hinausläuft. Nun wollen wir einmal davon ausgehen, daß die Arbeit im Editorfenster die wichtigere ist und dementsprechend die Prioritäten anders verteilen. Verlassen Sie den Editor durch Ctrl und X, und ändern Sie die Priorität des CLI auf 99 (*ChangeTaskPri 99*). Starten Sie erneut den Editor (*Run ED RAM:Unterschied*), und setzen Sie anschließend die Priorität des CLI-Prozesses auf -99. Schauen Sie sich mit *Status Full* die richtige Verteilung der Prioritäten an. Diese müßte etwa so aussehen:

```
Task  1: stk 1600, gv 150, pri 157 Loaded as command: status
Task  2: stk 3200, gv 150, pri  99 Loaded as command: ed
```

Lassen Sie sich von der eigenartigen Priorität 157 nicht verwirren. Diese müßte eigentlich -99 sein. Der Fehler liegt aber in der Anzeige des Befehls-Status. Dieser gibt nur Zahlen von 0-255 aus, die Priorität nimmt aber Werte von -128 bis +127 an. Daher wird eine -99 als +157 angezeigt (Das von uns abgedruckte Programm Task zeigt natürlich die richtigen Werte an.). Nun wollen wir den selben Test erneut durchführen, allerdings mit den geänderten Werten. Starten Sie also im CLI die Ausgabe eines längeren Directorys, und versuchen Sie, im Editor den Cursor zu bewegen.

Sie werden schnell feststellen, daß die Arbeit mit dem ED praktisch nicht mehr behindert wird. Allerdings hält die Ausgabe der Dateinamen auf dem Bildschirm fast völlig an, wenn Sie im Editorfenster eine Taste gedrückt halten. Jetzt wartet nicht mehr der ED auf Pausen im Directory, sondern CLI 1 auf Pausen des ED.

Fassen wir diese Informationen noch einmal kurz zusammen:

- Problemloses Arbeiten mit mehreren Prozessen erfordert meist eine richtige Verteilung der Prioritäten. Dann benötigen die

weniger wichtigen Aufgaben zwar etwas mehr Zeit, die wichtige Arbeit wird aber praktisch nicht behindert.

- Die gewünschte Verteilung der Wichtigkeit kann man mit dem Befehl ChangeTaskPri ändern. Dabei können Werte von -128 bis +127 benutzt werden. Obwohl in unserem Beispiel Werte von +99 bis -99 benutzt wurden, sollten Sie normalerweise nur Werte von +5 bis -5 einsetzen. Wir werden im nächsten Abschnitt noch genauer auf die Gründe eingehen. In unserem Beispiel waren die großen Werte allerdings problemlos möglich.

- Der Einsatz von ChangeTaskPri sollte richtig geplant werden. Ändern kann man immer nur die Priorität des CLI-Prozesses, von dem aus der Befehl aufgerufen wird. Allerdings gibt er seine Priorität ebenso an alle "Tochterprozesse" weiter (wie Standard-Ein-/Ausgabe und aktuelles Verzeichnis). Dadurch kann auch die Priorität von mit *Run* gestarteten Prozessen auf die gewünschten Werte eingestellt werden.

- Überprüfen kann man die Verteilung der Prioritäten durch *Status Full*. Allerdings werden negative Zahlen falsch ausgegeben. Den richtigen Wert erhalten Sie, indem Sie die Differenz von 256 und angezeigtem Wert bilden. Wird 255 angezeigt, so ergibt diese Differenz -1 den richtigen Wert.

- Einfluß können Sie mit ChangeTaskPri nur auf CLI-Prozesse nehmen. Damit ist es also nicht möglich, von der Workbench aus gestartete Programme in ihrer Wichtigkeit zu verändern. Im Kapitel 9 haben wir den neuen CLI-Befehl TaskPri abgedruckt, der nicht nur die Priorität von Workbench-Programmen, sondern die aller Tasks ändern kann: ein wichtiges Hilfsmittel zur völligen Kontrolle über das Multitasking des Amiga.

7.7 Was zu beachten ist

Wir haben schon zu Beginn dieses Kapitels darauf hingewiesen, daß Multitasking nicht nur gewichtige Vorteile hat, sondern

auch richtig eingesetzt werden will. In diesem Abschnitt wollen wir Ihnen nun einige Einschränkungen und Gefahren zeigen, damit Sie sich nicht "am heißen Kartoffelwasser die Finger verbrennen". Dabei wollen wir Ihnen gleich mit einem einfachen Beispiel zeigen, wo Probleme möglich sind.

```
CD DF0:
RUN DIR
RUN LIST
```

Wenn Sie die beiden letzten Zeilen relativ schnell schreiben und Return drücken, haben Sie besonders als Besitzer eines Amiga 500 ein beeindruckendes Geräuscherlebnis. Es hört sich etwa so an, als würde der Amiga jetzt die Diskette in zwei Hälften zersägen, um jedem Befehl eine Hälfte zur Verfügung stellen zu können. Das Problem liegt daran, daß jeder Prozeß, wenn er an der Reihe ist, wieder ein bißchen von Diskette lesen kann. Da die zu lesenden Bereiche oft weit auseinanderliegen, muß der Lesekopf der Floppy dabei große Strecken zurücklegen. Im extremsten Fall muß er für den einen Prozeß oben bei Spur 79 lesen und anschließend sofort für den anderen Prozeß auf Spur 0 zurück. Das belastet nicht nur die Mechanik und Ihre Ohren, sondern ist echte Zeitverschwendung. Manchmal kann das Abarbeiten dieser beiden Befehle deutlich länger dauern als die Ausführung hintereinander.

Unser Beispiel war natürlich nicht sinnvoll, weil die Ausgaben ja durcheinander in dem einen Fenster erfolgen. Es sollte nur das grundsätzliche Phänomen deutlich machen. Dieses Problem taucht aber praktisch ständig auf, wenn ein Prozeß Informationen von der Diskette liest und in einem anderen CLI der Befehl geladen werden muß. Das passiert in dem folgenden Beispiel:

```
RUN COPY DF0:Text PRT:
CD DF0:
```

Während *Copy* auf die Diskette zugreift, um den Text zu lesen, muß das CLI den nächsten Befehl (CD) ebenfalls von der Diskette lesen. Nur wird in diesem Fall das Hin- und Herfahren des Floppykopfes nicht so lange dauern. Abhilfe bietet hier eigentlich nur die Verwendung eines zweiten Laufwerkes. Ob das ein

externes Laufwerk oder die RAM-Disk ist, spielt dabei keine Rolle. Wichtig ist, daß möglichst nicht mehrere Prozesse gleichzeitig auf dasselbe Laufwerk zugreifen. In unserem Fall müssen also nur die CLI-Befehle in df0 und der Text in df1 sein, und schon tritt das Problem kaum noch auf.

Ein weiteres Problem taucht grundsätzlich auf, wenn mehrere Prozesse gleichzeitig auf eine Datei zugreifen wollen. Problemlos ist der gleichzeitige Lesezugriff auf die selbe Datei. Das können Sie leicht ausprobieren, indem Sie ein zweites CLI öffnen und von beiden CLI-Prozessen aus die Datei im jeweiligen Fenster ausgeben lassen (Type). Hat aber ein Programm oder Prozeß eine Datei zum Schreiben geöffnet, so verweigert AmigaDOS jedem anderen Prozeß den Zugriff auf diese Datei. Das ist auch sehr sinnvoll, da sonst leicht ungültige Daten gelesen werden könnten. Probieren Sie das einmal im folgenden Beispiel aus:

```
COPY DF0:S/Startup-Sequence RAM:Datei
CD RAM:
COPY Datei Datei1
```

Nun wollen wir versuchen, einerseits Datei1 neu zu beschreiben und gleichzeitig aus Datei1 zu lesen. Geben Sie dazu einfach ein:

```
RUN TYPE >Datei1 Datei
Type Datei1
```

Nach kurzer Zeit erscheint die Meldung "Can't open Datei1". Fragt man mit *Why* den Grund für den Fehler ab, so meldet AmigaDOS:

```
Last command failed because object in use
```

Während der erste Type-Befehl in die Datei1 schreibt, kann der zweite Type-Befehl nicht mehr daraus lesen. Dies gilt aber auch in umgekehrter Reihenfolge. Liest gerade ein Prozeß aus einer Datei und versucht ein anderer Prozeß die selbe Datei zum Schreiben zu öffnen, gibt es eine Fehlermeldung. So verursacht beispielsweise der Versuch, die Ausgabe in Datei1 zu lenken, während ein anderes CLI daraus liest:

```
TYPE >Datei1 Datei
```

eine Fehlermeldung "CLI error: Unable to open redirection file". Zwar taucht dieses Problem nicht allzu häufig auf, jedoch hat es eine besondere Bedeutung für die Programmierung von eigenen Programmen und Befehlen. Schreiben Sie beispielsweise ein BASIC-Programm, das eine vorhandene Datei zum Schreiben öffnet, und brechen Sie das Programm ab, ohne diese Datei wieder zu schließen, dann kann kein anderer Prozeß mehr auf diese Datei zugreifen. Glücklicherweise schließt AmigaBASIC alle offenen Dateien, wenn man die Arbeit damit beendet. Aber ein selbstgeschriebenes C-Programm sollte auf keinen Fall enden, ohne alle geöffneten Datein wieder ordnungsgemäß zu schließen.

Kommen wir nun zum letzten und wichtigsten Hinweis bei der Arbeit mit mehreren Prozessen. Wenn Sie die Prioritäten ändern, sollten Sie möglichst nicht die Werte <-5 oder >5 wählen. Denn durch einen Wert größer als 5 haben Sie beispielsweise eine höhere Priorität als das TrackDisk.Device, das für die Steuerung der Diskettenzugriffe zuständig ist. Das kann aber dazu führen, daß nicht sauber programmierte Programme mit dieser Priorität das ganze System abstürzen lassen. Dazu müssen Sie wissen, daß ein Programm in einem Multitasking-System Wartezeiten nicht durch Schleifen oder unsinnige Befehlen erreichen darf. Dafür gibt es fertige Routinen, die solche Wartezeiten möglich machen. Durch diese Routinen kommen dann aber andere Prozesse an die Reihe. Nehmen wir zwei Beispiele für dieses Phänomen, die Sie höchstens dann ausprobieren sollten, wenn Sie alle wichtigen Arbeiten gespeichert haben.

Achtung: Wenn Sie die folgenden Beispiele ausprobieren, verlieren Sie die Kontrolle über den Amiga. Speichern Sie daher alle Dateien und Inhalt der RAM-Disk.

1. Setzen Sie die Priorität des aktuellen CLI-Prozesses auf 50. Starten Sie AmigaBASIC mit Run. Klicken Sie in das linke Fenster und schreiben Sie *Text* hinein. Drücken Sie Return. Wahrscheinlich werden Sie plötzlich merken, daß keine Tastatureingaben mehr möglich sind. Wenn Sie Glück haben, können Sie wenigstens noch die Maus bewegen. Klicken Sie einfach in ein paar Fenster, und plötzlich steht auch die Maus. Dadurch, daß AmigaBASIC eine hö-

here Priorität als alle anderen Tasks hat (also auch als alle System-Tasks), können diese nicht mehr abgearbeitet werden. Da AmigaBASIC aber direkt auf deren Ergebnisse wartet, ohne die entsprechenden Warteroutinen des Betriebssystems zu benutzen, treten diese Ereignisse nie ein. Das System hängt, obwohl es nicht abgestürzt ist.

2. Für das zweite Beispiel benötigen Sie den neuen CLI-Befehl *TaskPri* aus diesem Buch. Starten Sie das Programm Lines aus dem Demo-Ordner. Geben ein:

```
TASKPRI Lines 50
```

Klicken Sie in das Lines-Fenster. Anschließend sind keine Tastatureingaben und keine Mausbewegungen mehr möglich.

Wir hoffen, daß Ihnen diese beiden Beispiele die Verantwortung verdeutlicht haben, die Sie durch eine Änderung der Prioritäten übernehmen. Deshalb noch einmal unser Rat: Arbeiten Sie im Normalfall nur mit Prioritäten von -5 bis 5, da dann die wichtigen System-Tasks in regelmäßigen Abständen abgearbeitet werden können.

8. AmigaDOS Intern

Natürlich macht es sehr viel Spaß, mit dem CLI zu arbeiten, aber nach einiger Zeit wird es Sie vielleicht doch interessieren, wie das CLI und AmigaDOS eigentlich funktionieren. Das ist besonders dann interessant, wenn Sie eigene Programme schreiben wollen. Dann können Sie sich an der internen Funktionsweise der vorhandenen CLI-Befehle viele Tips und Tricks zur eigenen Programmierung abgucken. Aus diesem Grund werden wir Ihnen in diesem Kapitel einiges über die interne Funktionsweise verraten, was wir in mühsamer Kleinarbeit herausgefunden haben. Viele dieser Informationen werden Sie in keinem anderen Buch finden, so beispielsweise eine Beschreibung des internen Aufbaus der CLI-Befehle und deren Nutzung der Global-Vektor-Table. Diese globalen Vektoren sorgen übrigens für die geheimnisvolle Anzeige "gv 150" nach einem Status Full.

Eins soll dieses Buch allerdings nicht leisten: komplette Informationen über den Amiga liefern. Deshalb werden wir auf Tasks und deren Strukturen, Message-Ports usw. nicht oder nur ganz kurz eingehen. Denn ein DATA BECKER Intern will dieses Buch nicht sein. Vielmehr wollen wir Ihnen einige äußerst interessante Informationen über das AmigaDOS weitergeben.

Um alle Informationen in diesem Kapitel verstehen und anwenden zu können, sollten Sie über C- und Maschinensprachekenntnisse verfügen. Wir werden aber versuchen, die Informationen so verständlich weiterzugeben, daß auch Amiga-Besitzer ohne Erfahrung mit diesen Programmiersprachen zumindest eine Vorstellung von den Abläufen bekommen.

8.1 DOS, Devices, Handler, Packets

Vielleicht haben Sie sich schon einmal gefragt, wie das AmigaDOS eigentlich grundsätzlich funktioniert, wie es beispielsweise ein File sowohl auf eine Diskette als auch in die RAM-Disk oder zum Drucker schicken kann. In diesem Abschnitt wollen

wir Ihnen kurz erklären, wie dieser Vorgang abläuft und was Devices und Handler damit zu tun haben.

DOS und Dateien

Angenommen, wir wollen eine Datei zum Lesen öffnen. Dann benutzen wir die Funktion der DOS.library OPEN(), die zwei Parameter benötigt: File-Name und Art des Zugriffs (Read/ Write). DOS untersucht nun diesen Filenamen, um herauszufinden, worauf zugegriffen werden soll. Dabei werden folgende Besonderheiten berücksichtigt:

- Das File heißt *. In diesem Fall trägt DOS ein, daß die Ein-/ Ausgaben einfach über das aktuelle CLI-Fenster ablaufen. Das bedeutet nichts anderes, als daß als Messageport für die Ein-/Ausgabe der Messageport des Con-Handlers benutzt wird.

- Das File heißt NIL. In diesem Fall trägt das DOS ein, daß es keine Routine für Ein-/Ausgaben gibt. Bei späteren READ()- oder WRITE()-Aufrufen wird dann auch nichts ausgegeben oder geholt - und das ist ja gerade die Wirkung von NIL:.

- Der File-Name enthält einen :. AmigaDOS prüft, ob sich vor dem Doppelpunkt ein Device befindet, das es in der Device-Liste entdecken kann, beziehungsweise ein Volume-Name, der "mounted" ist. Letzteres bedeutet bei einem Diskettennamen, daß die entsprechende Diskette in einem Laufwerk vorhanden ist. Ist dies der Fall, dann wird der Messageport dieses Devices eingetragen und für die späteren Ein-/Ausgaben benutzt. Ist der Handler für dieses Device noch nicht im Speicher, dann wird er geladen und gestartet.

Wird dieses Device oder Volume nicht entdeckt, dann muß es sich bei den Zeichen vor dem Doppelpunkt um einen Volume-Namen handeln (beispielsweise *Workbench V1.2*), zu dem es keine passende Diskette im Laufwerk gibt. In diesem Fall bittet AmigaDOS in einem höflichen Requester, dieses Volume in irgendein Laufwerk einzulegen. Ist dies geschehen, kann das DOS verfahren, als hätte es das Device sofort entdeckt.

Angenommen, auf die so geöffnete Datei erfolgt einige Zeit später ein Zugriff mit READ(). Dann schaut AmigaDOS in einem internen Puffer nach, ob dort noch Zeichen vorhanden sind. Ist dies nicht der Fall, wird eine Message an das Device geschickt, um den Puffer wieder mit Daten zu füllen. AmigaDOS greift also nicht selbst direkt auf die Diskette zu, sondern fordert die Daten vom Device an. Auch viele andere Aktionen des DOS werden in Wirklichkeit vom zugehörigen Device erledigt. Dies sind beispielsweise: Umbenennen der Diskette, Umbenennen von Dateien oder Verzeichnissen, Erzeugen eines neuen Directorys usw.

DosPackets

Wie verständigt sich nun das DOS mit einem Device? Dazu werden die sogenannten DosPackets benutzt. Über das genaue Aussehen können Sie sich im Include-File dosextens.h informieren. Wichtig ist der Eintrag dp_Type, denn hier wird die gewünschte Aktion als Zahl eingetragen. Die möglichen Aktionen und deren Zahlen finden Sie ebenfalls in dem angegebenen Includefile. Will das DOS beispielsweise eine Diskette umbenennen, dann schickt es ein Packet mit *dp_Type = RENAME_DISK (=9)* an das Device und wartet auf das sogenannte Replypacket, die Antwort des Devices. Dort kann das DOS feststellen, ob das Umbenennen geklappt hat oder ob ein Fehler aufgetreten ist.

Devices und Handler

Bisher haben wir immer davon gesprochen, daß DOS nach einem Device schaut und mit diesem Nachrichten austauscht. In diesem Zusammenhang war der Begriff Devices eigentlich nicht ganz korrekt, hat sich aber so eingebürgert, daß wir ihn ebenfalls vorläufig benutzt haben. Nun wollen wir aber genau beschreiben, mit wem DOS da eigentlich arbeitet. Es gibt nämlich Devices und Handler, und DOS arbeitet normalerweise nur mit dem jeweiligen Handler. Ein solcher Handler kümmert sich um eine Schnittstelle des Amiga. Der Port-Handler ist beispielsweise für die serielle und parallele Schnittstelle zuständig, der Con-Handler für Tastatur und Bildschirm und das FileSystem für die

Disketten. Zu jedem Handler gehören wiederum Devices, die der Handler zur Benutzung der Schnittstelle einsetzt. Der Port-Handler arbeitet mit den drei Devices: parallel.device, printer.device und serial.device, das FileSystem mit trackdisk.device und der Con-Handler mit dem console.device.

Grundsätzlich ist ein Device immer näher an der Hardware als der zugehörige Handler und kann nur weniger komplexe Aktionen durchführen. Betrachten wir das am Beispiel FileSystem (Handler) und trackdisk.device. Das Filesystem verarbeitet die "höheren" Diskettenstrukturen (Dateien und Verzeichnisse), während das trackdisk.device mit den "tieferen" Strukturen (Tracks, Sektoren) arbeitet. Betrachten wir zur Erläuterung noch einmal das Beispiel eines READ()-Zugriffs auf eine Datei, die vom DOS geöffnet wurde. Das DOS stellt nun irgendwann fest, daß im eigenen Puffer keine gültigen Daten mehr vorhanden sind. Also schickt es ein Packet an das zugehörige FileSystem und fordert weitere Daten an (*dn_Type = ACTION_READ*). Das FileSystem stellt nun fest, in welchen Sektoren die nächsten gültigen Daten dieser Datei liegen. Anschließend prüft es, ob diese Sektoren vielleicht schon in der Pufferliste vorhanden sind. (Das FileSystem puffert eine Anzahl Sektoren, die Anzahl kann durch AddBuffers erhöht werden.) Ist dies der Fall, so holt es die gewünschten Daten aus dem Puffer und kopiert sie in den vom DOS angegebenen DOS-Puffer.

Interessanter ist der Fall, wenn die Daten nicht im Puffer des FileSystems sind. Dann schickt es nämlich eine Message an das zugehörige trackdisk.device, damit dies die Sektoren in den Puffer des FileSystems holt. Das Trackdisk-Device liest intern stets einen ganzen Track. Daher schaut es nun erst einmal nach, ob sich die gewünschten Sektoren vielleicht im Trackpuffer befinden. Ist dies der Fall, so kopiert es aus diesem Trackpuffer die Sektoren in den Puffer des FileSystems. Andernfalls berechnet es den Track, auf dem die Sektoren liegen, und liest diesen in den Speicher.

Sie sehen, der Zugriff auf Diskettendaten geschieht über mehrere Stationen (DOS, FileSystem (Handler), trackdisk.device, Disketten-Hardware), und jede Station stellt einen weiteren

Schritt zur konkreten Hardware dar. Dies ist mit ein Grund dafür, daß die Diskettenzugriffe des Amiga recht langsam sind. Jeder puffert irgend etwas zwischen, durchsucht seinen Puffer nach den gewünschten Daten und schickt Nachrichten an andere Tasks. Natürlich gibt es noch weitere Gründe für die recht langsame Geschwindigkeit des Amiga beim Diskettenzugriff, beispielsweise die Art der Verteilung der Daten auf Diskette und das Nichtausnutzen einiger vorhandener Möglichkeiten der Amiga-Hardware durch das trackdisk.device (Index-Synchronisation).

Übrigens gibt es eine Ausnahme in dieser Aufgabenverteilung: DOS, Handler, Device, Hardware - die RAM-Disk der Version V1.2. Für diese gab es nur eine einzige Task RAM, die sowohl Handler als auch Device darstellte, aber nicht alle möglichen Aktionen unterstützte. Daher kann man beispielsweise diese RAM-Disk nicht formatieren. Bei der Version V1.3 gibt es nun eine resetfeste RAM-Disk namens CARD:, die aus einem RAM-Handler und dem ramdrive.device besteht. Auf Grund dieses Aufbaus können nun auch alle gängigen Aktionen (auch Formatieren) mit der neuen RAM-Disk ausgeführt werden.

Wenn Sie sich übrigens darüber informieren wollen, welche Handler und Devices es gibt, so finden Sie die Handler im logischen Verzeichnis L: und die Devices in DEVS:. Interessante Kombinationen können Sie sich im Verzeichnis DEVS: in der Mount-List anschauen.

8.2 Relokatible Programme, Segmente, BPTR-Pointer

Relokatible Programme

Da der Amiga ein Multitasking-Betriebssystem hat, dürfen die Adressen, an die verschiedene Programme geladen werden, nicht feststehen. Sonst könnte ein zweites Programm nicht geladen werden, weil an seinem Platz schon ein anderes Programm im Speicher liegt. Daher sucht das AmigaDOS beim Laden eines Programms einen entsprechend großen Speicher und paßt alle Adressen an diesen Speicherbereich an. Um diese Vorgänge muß

sich ein Programmierer normalerweise nicht kümmern, weil der C-Compiler oder Assembler die notwendigen Informationen mit auf der Diskette ablegt. Problematisch wird die unbestimmte Lage eines Programms nur für denjenigen, der fertige Programme analysieren und verstehen will. Lädt er nämlich ein Programm und untersucht es mit einem Monitor oder Debugger, so liegt es nach dem nächsten Laden an einer ganz anderen Stelle. Dadurch werden Notizen, die man sich gemacht hat, kaum noch verwertbar.

Eine große Hilfe stellt der PROFIMAT von DATA BECKER dar. Er enthält einen komfortablen Assembler, Debugger und Reassembler. Was ist nun der Vorteil dieses Systems? Sie laden das zu untersuchende Programm in den Debugger und stellen anschließend mit dem Reassembler eine Art Source-File von dem Programm her. Dieses können Sie nun für eigene Zwecke dokumentieren und gegebenenfalls erweitern. In den meisten Fällen kann es recht problemlos vom Assembler wieder in ein lauffähiges Programm übersetzt werden.

Segmente

Gerade weil das Betriebssystem des Amiga darauf ausgerichtet ist, mehrere Programme fast gleichzeitig laufen zu lassen, gibt es natürlich Schwierigkeiten mit dem freien Speicher. Dieser liegt gewöhnlich nicht an einem Stück, sondern ist in viele mehr oder weniger große Stücke geteilt, getrennt durch Bereiche, die von Programmen belegt sind. Angenommen, es sind noch 500 KByte Speicher freie, aber das größte zusammenhängende Stück ist nur 80 KByte groß, dann könnte normalerweise kein 90 KByte großes Programm geladen werden. Daher sieht es das Betriebssystem vor, ein Programm in viele kleine Segmente zu zerlegen. Diese können dann in kleine freie Speicherbereiche geladen werden und bilden zusammen das lauffähige Programm. Auch hier kümmert sich AmigaDOS darum, daß die Adressen der einzelnen Programmteile an die Lage der Segmente angepaßt werden. Allerdings muß die grundsätzliche Einteilung in Segmente vorher vom Programmierer mit Hilfe von Anweisungen an den Compiler oder Assembler erfolgen.

Für jedes geladene Programm legt AmigaDOS eine Verkettung der einzelnen Segmente an. Vor jedem Segment liegen zwei Langworte (BPTR-Pointer). Ein Langwort zeigt auf die Adresse des nächsten Segments, und das davorliegende Langwort gibt die Länge dieses Segments an. Mit Hilfe dieser Segmentliste ist es nun möglich, alle Programmteile eines Programms im Speicher zu finden und einzeln mit einem Reassembler in Source-Programme rückzuübersetzen.

BPTR-Pointer

Immer wieder werden Sie in C-Programmen, Include-Files oder Dokumentationen des Amiga-Betriebssystems auf die BPTR-Pointer stoßen. Was hat man sich darunter vorzustellen und wie geht man damit um? Entstanden sind die BPTR-Pointer auf Grund der Entwicklung des AmigaDOS in BCPL. In C zeigt ein Zeiger immer direkt auf die Speicherstelle, an der das Objekt liegt. Ein BPTR-Pointer ist ein solcher Zeiger geteilt durch vier. Zeigt ein Pointer also auf ein Objekt ab der Speicherstelle 10000, so zeigt der BPTR auf 2500. Dadurch geht keine Information verloren, denn BPTR-Pointer zeigen stets nur auf Objekte, die an geraden Langwortadressen liegen dürfen. Wenn man so eine Adresse durch vier teilt und anschließend wieder mal vier nimmt, kommt genau dieselbe Adresse heraus.

Diesen Sachverhalt wollen wir noch kurz an einem konkreten Beispiel deutlich machen. Wenn Sie eine Segmentliste haben, die die verschiedenen Segmente eines Programms verkettet, so sind die Zeiger auf das nächste Segment jeweils BPTR-Pointer. Nehmen Sie also diesen BPTR-Pointer und multiplizieren ihn mit 4, so erhalten Sie die Adresse des nächsten Segments und dort genau den BPTR-Pointer auf ein mögliches weiteres. Segmente dürfen nur auf geraden Langwortadressen liegen. Um also aus dem BPTR-Pointer die korrekte Adresse zu erhalten, müssen Sie den Wert mit 4 multiplizieren.

8.3 Das Programm CLI

Obwohl wir meist vom CLI als Benutzerschnittstelle einschließlich der CLI-Fenster und CLI-Programme im Ordner C: sprechen, gibt es auch noch das CLI, ein Startprogramm für diese Benutzerschnittstelle. Es befindet sich entweder im Systemordner oder direkt im Hauptverzeichnis der Workbench-Diskette. Mit Hilfe dieses Programms kann das erste CLI-Fenster geöffnet werden, falls die Startup-Sequence das einzige Fenster mit End-CLI geschlossen hat. Vielleicht interessiert es Sie, wie dieses Programm aufgebaut ist und wie man so ein Programm schreiben kann. Das wollen wir Ihnen in diesem Kapitel vermitteln. Die konkrete Beschreibung gilt nur für das mit V1.2 ausgelieferte Programm CLI, die entsprechende Version bei V1.3 ist grundsätzlich gleich aufgebaut, aber in einigen Details anders programmiert.

Das CLI-Programm ist im Gegensatz zu den meisten CLI-Befehlen in C geschrieben. Schaut man sich den Aufbau dieses Programms an, so führt es eigentlich erstaunlich wenig Aufgaben aus.

Aufbau des Programms CLI

Am Anfang steht der übliche Vorspann, den jedes C-Programm hat. Dort wird getestet, ob das Programm von der Workbench oder dem CLI aus gestartet wurde. Bei einem Start vom CLI aus wird die Kommandozeile ausgewertet und in die Parameter zerlegt.

Das eigentliche Programm Main() testet, ob schon ein CLI-Fenster existiert. Ist das noch nicht der Fall, so werden 10000 Bytes Speicher belegt, das Prompt wird auf %N> und das aktuelle Verzeichnis auf SYS: gesetzt.

In jedem Fall wird ein Fenster geöffnet. Dies erfolgt über die DOS-Funktion Open(), als Name wird *Con:0/50/640/80/New CLI Window* übergeben.

Dann wird die DOS-Funktion Execute aufgerufen. Diese erwartet drei Parameter: Kommando-String, Input- und Output-Handle. Übergeben wird als String ein Null-String, als Output-Handle eine Null und als Input-Handle das geöffnete Con-Fenster. Execute selbst setzt CIS (Command-Input-Stream) und COS (Command-Output-Stream) auf die übergebenen File-Handle, lädt und startet anschließend den CLI-Befehl C:Run. Dieser liest solange aus dem Con-Fenster, führt die Befehle aus und schreibt die Ergebnisse ins Con-Fenster, bis als Eingabe ein EndCLI erfolgt. Dann wird noch CIS und COS restauriert, und Execute kehrt zurück.

Das CLI-Programm schließt das Fenster, gibt gegebenenfalls den Speicher frei und endet ebenfalls.

Sie werden wohl mit uns der Meinung sein, daß das CLI-Programm erstaunlich wenig macht. Die eigentliche Arbeit wird vom CLI-Befehl *Run* erledigt. Viel erstaunlicher ist unserer Meinung nach aber die Tatsache, daß das CLI-Programm nicht ohne den CLI-Befehl *Run* lauffähig ist. Das können wir auch leicht an einem Beispiel beweisen:

Erstellen Sie im RAM das Verzeichnis C, und kopieren Sie nur den Befehl Assign in dieses Verzeichnis. Setzen Sie anschließend C: auf RAM.C

```
ASSIGN C: RAM:C
```

Nun starten Sie von der Workbench aus das CLI-Programm. Sie werden kurz das Con-Fenster erscheinen sehen, aber es verschwindet sofort wieder. Das liegt daran, daß C:Run nicht gefunden wurde. Löschen Sie also diesen Befehl nicht auf der Workbench-Diskette.

Mit diesem Wissen um die Funktion des CLI-Programms ist es nun sehr einfach, ein eigenes CLI-Programm zu schreiben. Die dazu notwendigen Schritte sind:

- Öffnen eines Con-Fensters
- Aufruf der DOS-Funktion Execute mit drei Parametern:
- Zeiger auf 0 für Execute-String in D1
- Zeiger auf Input-Handle des Con-Fensters in D2
- Zeiger auf 0 für Output-Handle in D3
- Schließen des Con-Fensters

Die notwendigen Maschinensprachebefehle (kein vollständiges Programm) sehen dazu so aus:

```
; ------------------ OpenCliWindow ---------------------
 OpenCliWindow:PEA $3ED ; Modus: Old_File
 PEA conwindow          ; "CON:0/50/640/80/New Cli Window
 JSR call_Open          ; DOS-Funktion OPEN aufrufen
 MOVE.L D0,D2           ; File-Handle von Open "CON:"
 ADDQ.L #8,A7           ; Stack restaurieren
 BEQ LC2C47E            ; Open ging schief -> kein Execute
 CLR.L -(A7)            ; = Output-Handle
 MOVE.L D2,-(A7)        ; = Input-Handle
 PEA executestring      ; $C2B8C0 = 0
 JSR call_Execute       ; Lädt "C:RUN"
                        ; Übergibt CIS und COS
                        ; Unload "Run", CIS/COS restaurieren
 RTS

conwindow: dc.b "CON:0/50/640/80/Mein Cli Window", 0
executestring: dc.b 0,0,0,0
```

Wie gesagt, das Programm ist nicht vollständig, denn es muß vorher beispielsweise noch die DOS-Bibliothek geöffnet werden. Aber trotzdem zeigt dieser kurze Programmausschnitt sehr deutlich, wie einfach ein eigenes CLI-Programm geschrieben werden kann. Leider wird dadurch auch völlig die Kontrolle an C:Run übergeben, wir können nur abwarten, bis *Run* ein *EndCLI* als Befehl liest, ausführt und wieder zurückkehrt.

8.4 Die interne DOS-Bibliothek

Haben Sie sich schon einmal C-Programme mit einem Disassembler angeschaut oder eigene Maschinenprogramme geschrieben? Dann wird Ihnen folgende Befehlsfolge immer wieder auffallen:

```
MOVE.L   ZeigeraufBibliothek,A6
JSR      OFFSET(A6)
TST.L    D0
BEQ      Fehler
```

Mit dieser Befehlsfolge können die Routinen der Betriebssystem-Bibliotheken (Librarys) aufgerufen werden. Schauen Sie sich aber einmal einen CLI-Befehl wie *Run* an. Dort werden Sie einen solchen Aufruf kaum finden. Statt dessen tauchen dort folgende Befehlsfolgen immer wieder auf:

```
LC188E8:MOVE.L D1,D6 ; rette Anzahl GLOBAL VEKTORS (=$95)
 ADDI.L #$32,D1      ; Addiere $32 für AllocMem (=$C7)
 SUBA.L A0,A0        ; Setze A0 auf 0
 MOVE.L $74(A2),A4   ; = Allocate Memory (MEMF_PUBLIC)
                       (Länge=D1),Return BPTR(D1)
 MOVEQ #$C,D0
 JSR (A5)            ; AllocateMem
 TST.L D1            ; Speicher erhalten für Vektortabelle?
 BEQ.L LC189DE       ; nein -> Ende
 ADD.L #$32,D1       ; ja, weiter
```

Lassen Sie sich von den Kommentaren ersteinmal nicht irritieren. Dies ist ein kurzer Ausschnitt aus dem CLI-Befehl Run. Interessant ist die Befehlsfolge:

```
MOVE.L OFFSET(A2),A4
JSR    (A5)
TST.L  D1
```

Interessanterweise werden Sie im Programm Run wohl verschiedene Offsets finden, jedesmal wird aber über JSR(A5) das gleiche Unterprogramm aufgerufen. Anscheinend greifen hier die CLI-Kommandos auf eine interne Bibliothek zu. Die BASIC-Adresse der Bibliothek muß in A2 vorhanden sein, in A4 wird dann die Adresse der Routine abgelegt, und die Unterroutine in A5 ruft diese Routine nach einigen Vorarbeiten auf. Daraus ergeben sich zwei interessante Konsequenzen:

- CLI-Befehle sind relativ kurz und kompakt, auf jeden Fall kürzer als vergleichbare C-Programme.

- CLI-Befehle benötigen mehr Informationen als ein C-Programm in den Registern des Prozessors, wenn sie gestartet werden, beispielsweise die Adresse der internen Bibliothek.

Da diese Adresse von der Workbench nicht übergeben wird, können CLI-Befehle auch nicht von der Workbench aus gestartet werden. Würden Sie beispielsweise dem Kommando *Run* ein Icon verpassen und dieses dann von der Workbench aus doppelklicken, so könnten Sie damit einen gefährlichen Guru erzielen, der auch von einem Debugger nicht mehr abgefangen werden kann.

Wie arbeiten nun die CLI-Befehle mit der Global-Vektor-Tabelle? Im Kapitel 7.5 werden wir zwar noch genauer darauf eingehen, aber an dieser Stelle ist schon folgendes interessant: Die meisten CLI-Kommandos kopieren zu Beginn die Tabelle in einen freien Speicherbereich und tragen einige weitere, eigene Routinen dort ein. Dadurch hat dann jeder CLI-Befehl seine eigene Tabelle. Die Standardtabelle enthält $96 Einträge ($00 - $95), was Sie auch am Kommentar der ersten Zeile des Programmausschnitts ablesen können.

Haben Sie sich schon einmal mit *Status Full* wichtige Informationen über die CLI-Prozesse anzeigen lassen? Dort taucht immer ein geheimnissvoller Eintrag GV auf, der normalerweise 150 beträgt. Das ist gerade der dezimale Wert für $96. Mit GV wird also zu jedem Prozeß die Größe seiner Global-Vektor-Tabelle angezeigt. Allerdings ist fraglich, warum dieser Wert überhaupt angezeigt wird, denn für Anwender ist er ziemlich uninteressant, und bisher haben wir auch noch nirgendwo Informationen über diese Tabelle gefunden. Übrigens heißt diese Tabelle wohl deshalb Global Vektor Table, weil sie allen CLI-Prozessen zur Verfügung steht, also allgemein "bekannt" ist. Übrigens kann die Mount-List ab V1.3 erheblich mehr Informationen zu einem Device enthalten, da der Befehl Mount geändert wurde und neue Schlüsselworte versteht. So können Sie beispielsweise

```
GlobVec = -1
```

eintragen und damit verhindern, daß der Handler eine Global Vector Table erhält. Dies ist sehr wichtig für in C geschriebene Handler, da diese sonst normalerweise abstürzen. Wenn Sie also beispielsweise die resetfeste RAM-Disk mounten und in der Mount-List kein Eintrag "GlobVec = -1" vorhanden ist, nimmt

Ihnen der Amiga das sehr übel und stellt die Zusammenarbeit ein.

Der von uns abgedruckte Teil des Programms Run ist die Vorbereitung für das Kopieren dieser Tabelle. Hier wird gerade der dafür notwendige Speicher besorgt.

Insgesamt erhalten die CLI-Befehle beim Aufruf noch wesentlich mehr Informationen in den Prozessorregistern. Dadurch können sie ohne großen Rechenaufwand auf wichtige Variablen zugreifen. Unsere Nachforschungen haben folgende Inhalte ergeben:

D0	Anzahl Parameterzeichen in Befehlszeile.
D2	Größe des Programm-Stacks.
A0	Zeiger auf Parameter-Zeichen.
A2	Zeiger auf interne DOS-Library.
A5	Zeiger auf Routine zum Aufruf der Funktionen.
A6	Zeiger auf Rücksprungroutine.

Wird beispielsweise der Befehl Run ohne weitere Parameter aufgerufen (wie es die DOS-Funktion Execute macht), so sieht die Übergabe in den Registern beispielsweise so aus:

```
;------------------- Beispiel-Register ------------------
; D0 = 00000001 -> Nur Befehlsname "Run"
; D1 = 00C5AA7C
; D2 = 00000FA0    = 4000 = Programm-Stack bei CLI-Aufruf
; D3 = 00000FA8
; D4 = 00000001 -> Länge des Parameter-Strings ?
; D5 = 0000003E
; D6 = 00309107
; D7 = 00C5A794

; A0 = 00C27411 -> Zeiger:1. Parameter hinter "run" (010A)
; A2 = 00C04CA0 -> Zeiger auf DOS-Library
; A3 = 00C6F52C
; A4 = 00C6BB48
; A5 = 00FF44B4
; A6 = 00FF44A8 -> Routine, die den Funktionsaufruf beendet
; A7 = 00C721C4 -> Rücksprungadresse
```

Interessant ist noch die Art und Weise, wie der Parameter-String abgelegt wird. Dies entspricht nicht der üblichen C-Konvention (Zeichenkette durch Null abgeschlossen), sondern im ersten Byte

steht die String-Länge, und dahinter folgen die Zeichen. In unserem Beispiel lautet der übergebene String:

```
$01,$0A (1 Zeichen, und zwar Zeilenvorschub)
```

Zeiger auf so abgelegte Strings werden als BSTR bezeichnet. BPTR sind Zeiger, die vor der Benutzung noch mit 4 multipliziert werden müssen.

8.5 Das Beispiel Run

In diesem Abschnitt wollen wir uns nun gemeinsam den Befehl Run anschauen. An diesem konkreten Beispiel können wir noch genauer zeigen, wie die DOS-Kommandos aufgebaut sind. Insbesondere wollen wir verdeutlichen, wie geschickt eigene Vektoren in die Kopie der Global Vektor Table (GVT) eingetragen werden können. Hierzu gibt es wiederum fertige Routinen in der GVT. Dadurch wird es Ihnen einerseits leichter fallen, andere CLI-Kommandos zu analysieren, und vielleicht können Sie sich für die eigene Programmierung von Programmen viele Kniffe abschauen. Auch für die konkrete Beschreibung von Run gilt wie beim Programm CLI: Bei neueren Versionen haben sich Details geändert und werden sich weiter ändern, die grundsätzliche Funktionsweise gilt für V1.2 und V1.3.

Grundsätzlich bestehen die meisten CLI-Befehle aus zwei Segmenten (siehe 7.1). Im ersten Segment werden dabei die Vorbereitungen für den eigentlichen Programmablauf durchgeführt:

- Zuerst prüft eine Schleife nacheinander alle Segmente des Programms auf die notwendige Größe der Global Vektor Table (GVT). Das letzte Langwort in den Segmenten gibt diese Größe an. Der vorgegebene Minimalwert ist dabei #$95 = 149, weil die im Betriebssystem vorhandene Tabelle 150 Vektoren enthält. Wird bei dieser Prüfung ein höherer Wert gefunden, so wird durch ihn die notwendige Größe festgelegt.

- Anschließend wird für die Tabelle der notwendige Speicher berechnet. Dabei werden #$32 Langworte für weitere Infor-

mationen addiert, und dieser Speicher wird per AllocMem belegt.

- Die GVT im ROM wird in den freien Speicher kopiert. Dazu wird die interne Routine der DOS-Library

```
MOVE.L $70(A2),A4     ; = Fill in a Global-Vektor-Table
MOVEQ #$C,D0
JSR (A5)              ; Fill the Table
```

benutzt. Diese wird zweimal aufgerufen, da die GVT an zwei verschiedenen Stellen liegt. Dann werden aus der CLI-Struktur der aktuellen Tasks 15 Langworte in den freien Speicher kopiert. Diese Struktur steht in der GVT ab $218 und wird auch in der neuangelegten Tabelle an derselben Stelle abgelegt. Anschließend wird nun für das eigene Programm ab dem zweiten Segment ebenfalls diese Routine aufgerufen. Die Routine "Fill in a global VektorTable" erwartet in D1 einen BPTR auf die Segmentliste des Programms und liefert in D0 eine -1 zurück, wenn alles geklappt hat, ansonsten eine 0.

Der Aufbau der Segmente muß dabei folgendermaßen sein: Das letzte Langwort des Segments enthält die maximale Größe der Tabelle und damit die größtmögliche Vektornummer. Davor steht der Offset des einzutragenden Vektors ab Segmentstart und davor die Vektornummer. Ein Offset von Null beendet diesen Vorgang. Das Ende des zweiten Segments von Run sieht beispielsweise so aus:

```
$00000000 $00000001 $00000024 $00000096
Ende      Vektornr. Offset   Max.Größe Tabelle
```

In diesem Fall wird also der Vektor 1 eingetragen, die Adresse der zugehörigen Subroutine beginnt bei Segment 2+$24 und die folgende $00000000 als Offset beendet dieses Segment. Da keine weiteren Segmente vorhanden sind, ist die Routine "Fill in a global VektorTable" beendet.

- Anschließend wird noch der Vektor 1 mit der Adresse einer Routine überschrieben, die im ersten Segment steht und deshalb nicht von der Kopierroutine in die Tabelle eingetragen werden konnte. Diese Routine gibt den Speicher der neuan-

gelegten Vektortabelle wieder frei und beendet den CLI-Befehl. Zum Schluß wird nun Vektor Null aufgerufen. Dieser zeigt gerade auf den Programmstart des zweiten Segments, so daß die dortigen Programmteile abgearbeitet werden. Übrigens beginnt der Programmcode des zweiten Segments und damit des neuen Vektors Null nicht ganz am Anfang, sondern bei $24, weil davor noch der Name des CLI-Kommandos steht. Das sieht also (bei V1.2) so aus:

```
;-------------------------------------
; Run Segment 2
Offset: $00
L00C1A610:DC.B $30,$39          ; "09"
DC.B $11,$52,$55,$4E            ; ".Run"
DC.B $20,$20,$20,$20
DC.B $20,$20,$20,$20
DC.B $20,$20,$20,$20
DC.B $20,$20,$00,$00            ;$20,$20
DC.B $07,$73,$74,$61            ; ".sta"
DC.B $72,$74                    ; "rt"
Offset: $24 ab hier steht der Code von Vektor 0
```

Übrigens beginnt der Code von Vektor 0 beim Run-Befehl der Version V1.3 schon bei $04, der Vorspann fehlt also. Die eigentlich wichtige Routine von Vektor Null ist ebenso wie beim CLI-Programm sehr kurz. Vorher erfolgen im wesentlichen Vorarbeiten wie der Aufruf von Input(), Output() und AllocMem(). Der entscheidende Teil sieht so aus:

```
LC1A860:MOVE.L $3C(A1),$68(A1)
 LEA $3F4(A4),A3
 MOVE.L A3,D4          ; hole Zeiger auf TASK-NODE-Namen
 LSR.L #2,D4           ; Wandle BPTR: TASK-NODE-Namen in Zeiger
 MOVE.L $34(A1),D3     ; hole Priorität des Prozesses
 MOVE.L #$320,D2       ; Größe des Stacks in Langworten
 MOVE.L $20(A1),D1     ; BPRT auf Segmentliste für Prozeß
 MOVEQ #$58,D0
 MOVE.L $84(A2),A4     ; Create a Process
 ;                       D1 = BPTR to SegList array for process
 ;                       D2 = Stack for prozess in long word
 ;                       D3 = Priority of Process
 ;                       D4 = BPTR to task node name
 JSR (A5)
 MOVE.L D1,$1C(A1)     ; Zeiger auf Prozeß Message Port
 TST.L D1              ; wurde Prozeß eingerichtet?
 BNE.S LC1A892         ; Nein-> "Can't Create Background Task"
 BRA.L LC1A9C0         ; ja, weiter, Speicher freigeben, Ende!
```

Der CLI-Befehl Run richtet also im wesentlichen eine neue Task mit dem Namen Background Task ein, den Sie auch begutachten können, wenn Sie unseren Befehl Task benutzen.

Wir hoffen, daß Ihnen diese Informationen beim Verstehen der CLI-Befehle helfen können. Wir meinen, daß sich durch dieses Verständnis erst die volle Leistungsfähigkeit und Vielfalt des AmigaDOS verstehen und in eigenen Programmen einsetzen läßt. Wenn wir endlich wissen, wie beispielsweise Assign arbeitet, werden wir in der Lage sein, bessere Befehle zu entwickeln, und damit die Arbeit der normalen Anwender mit AmigaDOS und CLI noch leistungsfähiger und angenehmer zu gestalten. Viele CLI-Befehle benutzen nämlich einfach die im Betriebssystem eingebauten Fähigkeiten und sind deshalb unglaublich kurz. Der Befehl AddBuffers beispielsweise nutzt eine Fähigkeit des File-Systems, den Cache-Speicher für das Zwischenpuffern der Sektoren zu vergrößern. Somit muß AddBuffers dem FileSystem nur ein Dospacket mit dem Befehl MORE_CACHE und der gewünschten Anzahl Sektoren schicken, den Rest erledigt das File-System selbst.

9. Neue CLI-Befehle

Das Amiga-Betriebssystem besitzt neben vielen anderen Vorteilen noch den, daß es fast beliebig erweiterbar ist. So können neue Librarys hinzugefügt und durch zusätzliche Devices kann der Amiga an neue Geräte angeschlossen werden. Dadurch, daß die CLI-Befehle nicht unerreichbar in das Betriebssystem integriert sind, sondern als kleine Programme auf der Workbench-Diskette vorliegen, können auch nachträglich noch neue Befehle hinzugefügt werden. Als angehender CLI-Profi ist es Ihnen sicher längst bekannt, wo man auf der Workbench-Diskette nach den CLI-Befehlen zu suchen hat. Richtig, in dem Verzeichnis C. Das DOS findet die Befehle dort, weil als letztes immer das Directory mit dem Namen C: durchsucht wird. Beim Start des Rechners wird diesem Directory aber der Ordner C der Workbench-Diskette zugeordnet. Über diesen kleinen Umweg kann nun dort jeder CLI-Befehl gefunden werden.

Nirgendwo wird aber die Anzahl der Befehle in irgendeiner Weise eingetragen oder gar begrenzt. Sie könnnen sich Ihren Lieblings-CLI-Befehl dort so oft unter verschiedenen Namen hineinkopieren, bis der unliebsame Disk Full-Requester auf der Bildfläche erscheint. Eine etwas ernsthaftere Anwendung für diese Möglichkeit besteht darin, häufig aufgerufene Befehle noch ein zweites Mal dort abzuspeichern, jedoch unter einem einprägsamen kürzeren Namen. Beispiel: X für Execute, FC für FileCopy etc. (Als Besitzer der neuen Workbench V1.3 haben Sie natürlich noch die viel komfortablere Möglichkeit des ALIAS-Befehls. Zusätzlich wird dabei kein Speicherplatz auf der Diskette durch Anlegen von Kopien verschwendet). Es besteht auch die Möglichkeit, völlig neue Befehle in den C-Ordner einzutragen. Da ein Befehl ja im Grunde genommen nichts anderes ist als ein Programm, können Sie hier auch z.B. die Clock hineinkopieren. Bei der Ausgabe des Verzeichnisses C mit Hilfe des List-Befehls würde dieses Programm jedoch wegen seines hohen Speicherbedarfs aus dem Rahmen fallen. Die am häufigsten benutzten CLI-Befehle haben eines gemeinsam: Sie sind relativ kurz und können daher schnell in den Rechnerspeicher geladen

werden. Ein BASIC-Compiler wird sich daher weniger zur Erstellung eigener Befehle eignen.

Solch einen kompakten Code, wie ihn die echten CLI-Befehle enthalten, kann man auf dem Amiga höchstens unter Verwendung eines Assemblers erzeugen. Die Muttersprache der Befehle BCPL wird auf dem Amiga nicht angeboten. Eine echte Alternative stellt aber hier die Sprache C dar, die sich aus BCPL entwickelt hat und mit der auch große Teile des Betriebssystems programmiert worden sind. Eine CLI-typische Programmstruktur kann jedoch mit der Sprache C aus bestimmten Gründen, auf die hier nicht näher eingegangen wird, nicht erreicht werden. Bei der Entwicklung eigener Befehle sollte man aber die CLI-Umgebung durch einen angepaßten Programmierstil berücksichtigen. Zumindest auf die folgenden Besonderheiten sollte man achten:

1. Alle CLI-Befehle sind "nicht-interaktiv", das heißt, sie benötigen nach dem Start keine Informationen mehr vom Benutzer. Eventuelle Parameter müssen dem Befehl beim Aufruf direkt mitgegeben werden, indem sie in Form einer Liste direkt dahinter geschrieben werden (z.B. copy texte ordner).

2. Die Befehle geben im Normalfall ihre Informationen immer in dem CLI-Fenster aus, von dem aus sie aufgerufen worden sind. Niemals wird man es erleben, daß bei einem normalen Aufruf ein eigenes Fenster für Ausgaben erscheint.

3. Die echten CLI-Befehle beinhalten eine Hilfsfunktion für den Benutzer, die durch die Eingabe eines Fragezeichens aktiviert wird. Sollte der Anwender dem Befehl falsche Parameter mitgeben, so wird auch hierauf mit einer passenden Bemerkung reagiert. Beispiele: Hilfsfunktion:

```
Eingabe:    date ?
Ausgabe:    TIME,DATE,TO=VER/K:
```

Sinnvolle Fehlermeldung:

```
Eingabe:    date heute
Ausgabe:    ***Bad args
            - use DD-MMM-YY or <dayname> or yesterday etc.
            to set date, HH:MM:SS OR HH:MM to set time
```

9.1 Die Realisierung in C

Bei richtiger Programmierung wird ein Außenstehender einen in C geschriebenen Befehl nicht von einem echten CLI-Kommando unterscheiden können. Auch in dieser Sprache ist es möglich und auch üblich, die Parameter einem Programm direkt mitzugeben. Sogar die Umlenkung der Ausgaben mit Hilfe des Zeichens > ist möglich.

Auch wenn Ihr neuer Befehl zur Ausführung keine weiteren Angaben benötigt, so sollte doch die Eingabezeile ausgewertet werden. Falls der Anwender dennoch Parameter angibt, so beweist dies seine Unkenntnis über die Aufgabe dieses Kommandos. In diesem Falle sollte die Abarbeitung sicherheitshalber unterbleiben.

Auf dem Amiga wird die Auswertung der Eingabezeile in C folgendermaßen programmiert: Der zuerst aufgerufenen Main-Funktion werden die Eingabedaten automatisch in Form von zwei Parametern übergeben: Der erste Parameter, der meist den Namen ArgC (von Argument Counter) bekommt, ist vom Typ Int und enthält die Anzahl der übergebenen Argumente. Dabei wird der Name des Programms selber mitgezählt, so daß hier mindestens eine Eins enthalten ist. Der zweite Parameter bekommt gewöhnlich den Namen ArgV (von Argument Vector). Er muß deklariert werden als ein Feld von Zeigern auf den Datentyp Char. Die Elemente dieses Feldes enthalten dann Zeiger auf die Zeichenketten der Eingabezeile, die jeweils mit einer Null abgeschlossen sind.

Diese Aufbereitung übernimmt ein Programmteil, der durch den Linker bei der Erstellung des Programms in jedem Fall hinzugefügt wird. Diese Aufgabe wird also nicht, wie man viel-

leicht meinen könnte, vom CLI erledigt. Die ersten Zeilen einer Main-Funktion sehen daher meist so aus:

```
main(argc, argv)
int argc;
char *argv[];
{
...
```

Da ArgC bzw. ArgV von außen der Funktion Main übergeben werden, müssen sie noch vor der geschweiften Funktionsklammer als Parameter des entsprechenden Typs deklariert werden.

Ein kleines Beispiel soll Sie mit dieser Programmiertechnik vertraut machen. Wir haben alle C-Programme mit einem Aztek-Compiler übersetzt. Die Benutzung eines anderen Compilers dürfte aber keine Schwierigkeiten bereiten, wenn man die im zugehörigen Handbuch beschriebenen Besonderheiten beachtet.

```
/* Programm: Auswertung */

main(argc, argv)
int argc;
char *argv[];
   {
   int i;
   printf (" Anzahl: %d \n", argc);
   for ( i = 0; i < argc; i++)
      {
      printf (" Nr.: %d, Argument: %s \n",i , argv[i]);
      }
   }
```

Ruft man dieses Programm vom CLI aus auf durch:

```
Auswertung ein nochein undnochein Parameter
```

erhält man die folgende Ausgabe:

```
Anzahl: 5

        Nr.: 0 , Argument: Auswertung
        Nr.: 1 , Argument: ein
        Nr.: 2 , Argument: nochein
        Nr.: 3 , Argument: undnochein
        Nr.: 4 , Argument: Parameter
```

Ein Leerzeichen bei der Eingabe wird also als Trennzeichen der einzelnen Parameter interpretiert. Wie kann man dann aber einem C-Programm vom CLI aus einen Text übergeben, der auch Leerzeichen enthält? Natürlich wäre es möglich, aus den einzelnen Bruchstücken, die man ja in C in einem solchen Fall normalerweise erhalten würde, durch eine kleine Funktion wieder den Gesamttext zusammenzubasteln. Die einfachere Methode dürfte Ihnen mittlerweile aber auch geläufig sein: Der Text muß nur in Anführungszeichen gesetzt werden, um den gewünschten Erfolg zu erzielen. Der Aufruf unserer Beispielfunktion durch:

```
Auswertung "ein nochein undnochein Parameter"
```

hätte zum Ergebnis:

```
Anzahl: 2
Nr.: 0 , Argument: Auswertung
Nr.: 1 , Argument: ein nochein undnochein Parameter
```

Bis dahin stimmte alles mit der Parameterübergabe der echten CLI-Befehle überein. Nun könnte man sich die Frage stellen, ob auch die Sonderzeichen > bzw. < als ganz normale Zeichen an unser Programm übergeben werden oder ob sie etwa die bekannte Funktion besitzen, Ausgaben bzw. Eingaben des Programms auf beliebige Geräte umzuleiten. Dazu ein erster Test:

Durch den Befehl

```
Auswertung >df0:Datei Parameter
```

wird die Ausgabe tatsächlich in die gewünschte Datei umgeleitet. Es erscheint also nicht etwa die Ausgabe

```
Anzahl: 2
Nr.: 0 , Argument: >df0:Datei
Nr.: 1 , Argument: Parameter
```

auf dem Bildschirm. Der Versuch, die Eingabe mittels des Zeichens < umzuleiten, schlägt leider fehl. Die Ursache ist in dem C-Programmteil zu suchen, das die Auswertung der Eingabezeile vornimmt. Dabei wird nicht beachtet, welches Eingabegerät gerade aktuell ist, sondern grundsätzlich die Zeile von der Tastatur eingelesen. Trotzdem ist es möglich, auch von anderen Geräten

die Daten für die Eingabe zu empfangen. Das folgende Beispielprogramm demonstriert, wie man dafür vorzugehen hat:

```
#include <stdio.h>
FILE * Input();    /* Deklaration einer externen Funktion */
main(argc, argv)
int argc;
char *argv[];
   {
   int i;
   FILE *Eingabe;  /* Pointer auf Struktur-Typ FILE */
   char buffer[100];
   long Laenge;
   printf (" Anzahl Parameter: %d\n", argc);
   for ( i = 0; i < argc; i++)
       printf (" Argument: %s\n", argv[i]);
   Eingabe = Input();
   Gelesen = Read (Eingabe, &buffer[0], 30L);
   buffer [Laenge] = 0;
   printf (" Gelesen: %s\n", &buffer[0]);

   }
```

Erzeugt man mittels

```
echo >datei "eins zwei drei"
```

eine kleine Eingabedatei und startet nun das obige Programm durch

```
Eingabe: <datei hallo
```

erscheint auf dem Bildschirm die Ausgabe:

```
Anzahl Parameter: 2

Argument: Eingabe
Argument: hallo
Gelesen: eins zwei drei
```

Die Funktion von ArgC bzw. ArgV bleibt unverändert erhalten. Die Umleitungsangabe <datei wird jedoch vollständig aus den Angaben herausgefiltert, weswegen ArgC im obigen Beispiel auch nur die Anwesenheit zweier Argumente bestätigt.

Nach der Ausgabe der normalen Parameter wird durch die Funktion Input(), die sich in der DOS-Library befindet, das Standardeingabegerät des aufrufenden Programms (also das des

CLI) für unser Programm bereitgestellt. Die Aufgaben der hierbei benutzten Variablen sind:

Eingabe	**Zeiger auf Standardeingabe.**
Buffer	**Bereich, der die gelesenen Zeichen enthält.**
Laenge	**Tatsächliche Anzahl gelesener Zeichen.**

Durch das Kommando < in der CLI-Zeile wird die Standardeingabe auf das dort angegebene Gerät umgeschaltet. Dies wird übrigens automatisch vom AmigaDOS erledigt. Auch die bestehenden CLI-Befehle "sehen" nur die Argumente in der Kommandozeile, die keine Umlenkungszeichen enthalten, denn diese werden schon vorher ausgewertet.

9.2 Task - Die Aufgaben des Amiga

Wahrscheinlich hat es Sie schon immer gereizt zu wissen, was der Amiga wirklich macht, wenn Sie ihn nicht mit Aufgaben ausgelastet haben, Zwar gibt es nur den Befehl Status vom CLI aus, aber dieser zeigt Ihnen leider nur alle CLI-Prozesse und ihre Aufgaben, nicht aber die von der WB gestarteteten Programme und die internen Tätigkeiten des Amiga. Dabei wäre es doch von echtem Vorteil, wenn man sehen könnte, welche Aufgaben der Amiga wirklich intern erledigt. Nur dadurch hat man überhaup eine Möglichkeit, Einfluß auf diese Aufgaben zu nehmen. Deswegen haben wir für Sie zwei neue CLI-Befehle entwickelt: Task, um sich alle internen Aufgaben anzeigen zu lassen, und TaskPri, um entscheidenden Einfluß auf diese Aufgaben zu nehmen. Beginnen wir also mit Task. Dieser Befehl zeigt Ihnen nicht nur alle aktiven Tasks an, sondern gibt gleichzeitig auch noch ihre Priorität aus, so daß Sie die interne Bedeutung an der Priorität leicht ablesen können.

Bevor wir den Befehl als C-Programm abdrucken, wollen wir Ihnen zumindest einige Informationen über die grundsätzliche Funktionsweise geben. Der Amiga hat ein Betriebssystem, das völlig relativ ist. Keine Adresse ist sicher; alles kann an einer beliebigen Stelle liegen. Das zeigt sich schon durch die relokatiblen Programme, die an jede Adresse geladen werden können. Eine einzige feste Adresse gibt es allerdings, es ist die

Startadresse der sogenannten Execbase. Dies ist sozusagen die Startadresse der Grundtabelle des Amiga-Betriebssystems, und diese Adresse finden Sie stets in Speicherstelle $000004. In dieser Speicherstelle steht allerdings nur die Adresse der Execbase. Diese selbst liegt schon bei einem Amiga 500 mit Speicher-Erweiterung an einer andereren Stelle als bei einem Amiga ohne. Innerhalb dieser Execbase gibt es eine Fülle von wichtigen Zeigern, beispielsweise auf alle Devices, Librarys usw. Unter anderem findet man hier auch die Adressen der Listen, in denen die Tasks verwaltet werden. Grundsätzlich gibt es drei verschiedene Statusmöglichkeiten für jede Task:

Running	**Task ist gerade aktiv, wird abgearbeitet.**
Ready	**Task kann demnächst bearbeitet werden.**
Waiting	**Task wartet auf Ereignis, vorher wird er nicht abgearbeitet.**

Es ist natürlich immer nur eine Task im Zustand Running, während beliebig viele Ready oder Waiting sein können. Tritt das Ereignis ein, auf das eine Task in der Waiting-Liste wartet, so wird er automatisch vom Betriebssystem in die Ready-Liste überführt. Der Aufbau dieser Listen ist nicht ganz einfach, und wir wollen deshalb auch nicht weiter darauf eingehen. Wenn wir im folgenden sagen: Task zeigt alle Tasks an, so stimmt das nicht ganz. Es werden nur die im Zustand Waiting angezeigt. Das bedeutet aber kaum eine Einschränkung, da nur eine Task Running ist und selten mehrere Tasks Ready. Sollten Sie an weiteren Informationen interessiert sein, so schauen Sie bitte in das "Amiga Intern". Wir drucken im folgenden erst einmal den neuen CLI-Befehl ab.

```
#include <exec/types.h>
#include <exec/execbase.h>
#include <exec/tasks.h>
#include <exec/exec.h>
#include <exec/execname.h>
#include <exec/lists.h>
extern struct ExecBase *SysBase;
main()
{
    struct Task *task;
    char *names [20];
    int pri [20];
    int count,i;
    count = 0;
    Disable();
```

```
    for (task = (struct Task *)SysBase->TaskWait.lh_Head;
    task->tc_Node.ln_Succ; /* Ende, wenn Zeiger auf nächsten
                              Eintrag = 0 */
   task = (struct Task *)task->tc_Node.ln_Succ)
      {
         names[count] = task->tc_Node.ln_Name;
         pri[count++] = task->tc_Node.ln_Pri;
      }
    Enable();
    for (i = 0; i < count; i++)
      {
         printf ("%s",names[i]);
         printf ("\r\t\t\t%d\n",pri[i]);
      }
}
```

Zur Funktionsweise des Programms

Zuerst einmal werden alle notwendigen Header-Files in den Source eingebunden, und ein Pointer mit dem Namen SysBase definiert, der auf die oben beschriebene ExecBase zeigt. Nun müssen erst einmal die nötigen Variablen definiert werden. Zuerst benötigen wir einen Zeiger auf die Struktur Task. Die von uns gesuchten Strukturen müssen wir nicht einrichten, da sie schon in den Task-Listen vorhanden sind. Weil sich der Inhalt dieser Task-Strukturen ständig ändert, können wir nicht in aller Ruhe die Namen auf dem Bildschirm ausgeben, sondern müssen deren Adressen erst einmal möglichst schnell in ein eigenes Feld retten. Besonders nützlich ist dabei die Funktion Disable(), die das sogenannte Task-Switching sperrt. Von diesem Augenblick an sind wir die einzige aktive Task, und die Listen werden nicht mehr geändert. Wir können also in aller Ruhe die Liste auslesen, bis wir durch eine Null ihr Ende feststellen:

```
task->tc_Node.ln_Succ; /* Ende, wenn Zeiger auf nächsten Eintrag = 0 */
```

Anschließend können wir das Task-Switching durch Enable() wieder einschalten und in Ruhe unsere gewonnene Liste ausgeben. Dabei gehen wir natürlich das Risiko ein, daß sich die Namen in der Liste mittlerweile verändert haben, weil wir lediglich die Zeiger auf diese Namen retten. Ansonsten hätten wir erst mit der Kopierfunktion Strncpy die einzelnen Namen retten müssen, was wesentlich mehr Zeit und Speicher in Anspruch ge-

nommen hätte. Außerdem sind maximal 20 Einträge möglich, was in den meisten Fällen jedoch völlig ausreicht.

Von einem dritten CLI aus gestartet, könnte eine Ausgabe unseres Programms folgendermaßen aussehen:

```
Background CLI          0
CON                     5
CON                     5
New CLI                 0
New CLI                 0
Workbench               1
File System            10
input.device           20
trackdisk.device        5
File System            10
trackdisk.device        5
CON                     5
```

Die Task mit dem Namen Background CLI ist in diesem Fall verantwortlich für das Programm Textomat, das vom CLI aus durch Run gestartet worden ist. Die insgesamt drei Tasks mit dem Namen Con kümmern sich um die Auswertung von Fensterein- und ausgaben, darunter auch die des Textomat-Programms. Die Tasks New CLI stammen von weiteren mit NewCLI gestarteten CLI-Prozessen. Interessanterweise richtet auch die beliebte Workbench eine Task ein und reiht sich damit in die vielen Aufgaben ein, die der Amiga ständig zu erledigen hat. Der Kenner schließt aus den restlichen Angaben, daß hier ein Amiga mit zwei Diskettenlaufwerken am Werk gewesen sein muß, da jeweils zwei Tasks mit dem Namen File System bzw. Trackdisk.device vorhanden sind.

9.3 TaskPri - Die Aufgabenverteilung beliebig ändern

Dieser neue CLI-Befehl ist eine Weiterentwicklung des schon in Kapitel 7.6 besprochenen Kommandos ChangeTaskPri. TaskPri kann jedoch mehr: Ein großer Nachteil von ChangeTaskPri liegt darin, daß immer nur die Priorität des gerade aktuellen CLI geändert werden kann. Es kann also nicht von einem CLI aus die Priorität eines anderen beeinflußt werden. TaskPri ermöglicht es sogar, nicht nur die Priorität beliebiger Prozesse, sondern

die aller vorhandenen Tasks zu manipulieren, die ja durch den schon oben behandelten Befehl Task angezeigt werden. Da dieser Befehl Eingaben des Benutzers auswerten muß (den Namen und die neue Priorität einer Task), ist er ein wenig umfangreicher geworden. Hier nun erst einmal das C-Listing unseres neuen CLI-Befehls:

```
#include <exec/types.h>
#include <exec/execbase.h>
#include <exec/tasks.h>
#include <exec/exec.h>
#include <exec/execname.h>
#include <exec/lists.h>
extern struct ExecBase *SysBase;
long FindTask();
int settaskpri();
main(argc, argv)
int argc;
char *argv[];
   {
   struct Task *MeinTask;
   long pri;
   int ergebnis;
   char oldpri;
   long task;
   pri = 0L;
   if (argc != 3)
      {
      printf(" taskname priorität\n");
      exit(FALSE);
      }
   ergebnis = Value(argv[2], &pri);
   if (ergebnis == FALSE)
      {
      printf (" 128 <= Taskpri <= 127 nicht: %ld\n", pri);
      exit(FALSE);
      }
   task = FindTask (argv[1]);
   if (task == 0)
      {
      printf ("Task <%s> nicht gefunden !\n", argv[1]);
      exit(FALSE);
      }
   oldpri = ( char ) SetTaskPri (task, pri);
   printf ("Alte Priorität war: %d  neue ist %ld\n", oldpri, pri);
   exit (TRUE);

}

/* Aus bis zu drei Ziffern Zahl 0 - 255 machen */
Value(string, wert)
```

```
char *string;
long *wert;
   {
   int i;
   int anzziffern;
   char zeichen;
   int ziffer;
   int zehner;
   zehner = 1;
   anzziffern = 0;
   if (*string == '-')     /* negative Zahl */
      {
      zehner = -1;         /* in negative Zahl wandeln */
      string++;            /* Minuszeichen überlesen */
      }
      for (i = 1; i <= 4; i++)   /* Anzahl Ziffern bestimmen */
      {
      if (*string == 0)
         break;
      string++;
      anzziffern++;
      }
   if ((anzziffern == 0) || (anzziffern > 3)) /* nur drei Ziffern
                                                 erlaubt */
      return (FALSE);
   for (i = anzziffern; i > 0; i-- )          /* Zahl bilden */
      {
      string--;
      zeichen = *string;        /* String jetzt von hinten lesen*/
      if (zeichen < '0' || zeichen > '9')
         return (FALSE);
      ziffer = zeichen - '0';
      *wert = *wert + ziffer * zehner;
      zehner = zehner * 10;
      }
   if (*wert < -128 || *wert > 127)
      return (FALSE);
   return (TRUE);
   }
```

Zur Funktion des Programms

Die ersten sieben Zeilen des Listings stimmen mit dem des Programms Task überein. Anschließend wird die externe Funktion FindTask() deklariert, die sich in der Exec-Library befindet. Ihre Aufgabe ist es, die Task-Listen nach einem Eintrag mit einem vorgegebenen Namen abzusuchen. Der Funktion muß hierzu als Parameter ein Zeiger auf den gesuchten Task-Namen mitgegeben werden. Wird ein passender Eintrag gefunden, so erhält man als Ergebnis einen Zeiger auf die entsprechende Task-

Struktur. Im Gegensatz zu vielen anderen Funktionen auf dem Amiga legt die FindTask-Funktion Wert auf Groß- und Kleinschreibung. Wird also der Name einer Task nicht genau so übergeben, wie ihn unser Task-Befehl ausgibt, findet die FindTask-Funktion einen solchen Eintrag einfach nicht.

Problematisch wird es auch, wenn mehrere Tasks den gleichen Namen besitzen. In einem solchen Fall wird nur ein Zeiger auf diejenige Struktur zurückgegeben, die FindTask zuerst unter diesem Namen findet. Dieses Problem kann jedoch nur durch einen erheblich größeren Programmieraufwand beseitigt werden.

Als zweite externe Funktion wird SetTaskPri() deklariert. Der Name beschreibt schon sehr gut die Aufgabe dieser Funktion: Die Priorität einer Task wird auf einen beliebigen neuen Wert gesetzt. Die Funktion erwartet als Parameter natürlich die neue Priorität (-128 bis +127) und den Strukturzeiger, den man über FindTask erhalten hat. Der Rückgabewert enthält dann die bisherige Priorität dieser Task.

Die Main-Funktion von TaskPri ist aufgrund der konsequenten Nutzung der vorhandenen Betriebssystemfunktionen einfacher zu verstehen als die des zuletzt besprochenen Task-Befehls. Die Auswertung der Übergabeparameter ist genauso programmiert, wie es zu Beginn dieses Kapitels beschrieben worden ist. Diesmal ist auch ein kleiner Syntaxcheck vorhanden, der bei Fehleingaben eine entsprechende Meldung ausgibt: Sollten dem Befehl nicht genau zwei zusätzliche Parameter mitgegeben werden (argc !== 3), so wird die Main-Funktion nach der Ausgabe von Taskname Priorität vorzeitig verlassen. Diese Mitteilung erscheint somit auch bei der Eingabe eines einzelnen Fragezeichens. Anschließend werden die eingegebenen Parameter durch die Funktion Value() auf das Long-Format gebracht, denn mit ASCII-Werten kann SetTaskPri wirklich nichts anfangen. Value() erwartet als Parameter einen Zeiger auf die Eingabedaten (argv[2]) und die Adresse einer Variablen, in die das konvertierte Ergebnis abgelegt werden kann (&pri). Sollte die Funktion unpassende Werte vorfinden, gibt Value() ein False zurück, und man wird über die erlaubten Parameter informiert. Der Rest der Main-Funktion erklärt sich unter Berücksichtigung der obigen

Informationen fast von selbst: Falls die gewünschte Task-Struktur gefunden wird - FindTask liefert einen Wert ungleich null - wird hier die neue Priorität eingetragen und die bisherige im CLI-Fenster ausgegeben.

Bei der Konvertierungsfunktion Value() handelt es sich mehr um ein reines C-Problem, weswegen hier nicht näher auf die Funktionsweise eingegangen wird. Die Aufgabe der Funktion besteht einfach darin, die ASCII-Eingabe - falls dies überhaupt möglich ist - in einen Zahlenwert zwischen -128 und 127 umzuwandeln.

Zur Bedienung des Programms

Der Aufruf unserer neuen CLI-Funktion ist einfach: Suchen Sie sich aus der Liste, die der Befehl Task ausgibt, einen Namen aus, und legen Sie für ihn eine neue Priorität fest. Mit diesen beiden Angaben wird dann TaskPri aufgerufen. Sollte der Task-Name Leerzeichen enthalten (z.B. File System), muß er in Anführungsstriche gesetzt werden.

Die Wirkung dieses Befehls kann man sehr schön an dem nun folgenden Beispiel beobachten: Lassen Sie einmal das Inhaltsverzeichnis der Workbench-Diskette durch *Dir* auf dem CLI-Fenster ausgeben. Bewegen Sie dabei gleichzeitig den Mauspfeil auf dem Bildschirm umher. Ergebnis: Die Maus läßt das völlig kalt; die Bewegung des Pfeils auf dem Monitor ist ganz normal. Geben Sie nun einmal vom CLI aus ein: *TaskPri Input.device -1.* Als Antwort erhalten Sie dann: "Alte Priorität war: 20, neue ist -1". Wiederholen Sie nun das Spielchen mit der Maus. Sie werden sehen, daß nun der Pfeil bei der Ausgabe des Verzeichnisses ganz schön ruckelt. Das Input-Device kommt eben ab und zu einfach nicht mehr dazwischen.

Nun, sehr sinnvoll ist das Beipiel ja nicht gerade, dafür zeigt es aber sehr deutlich, was für ein mächtiges Werkzeug der neue CLI-Befehl darstellt. Natürlich gibt es für ihn auch vernünftige Anwendungen:

Stellen Sie sich nur einmal vor, Sie möchten, während Sie in einem Textverarbeitungsprogramm z.B. einen Brief schreiben, von

einem zweiten CLI aus einen längeren Text auf den Drucker ausgeben. In einigen Fällen wird hierdurch die Geschwindigkeit des Textverarbeitungsprogramms dermaßen stark beeinträchtigt, daß damit kaum noch gearbeitet werden kann. Hier hilft es nun, wenn die Priorität des Printer.device z.B. auf -5 und die der Textverarbeitung auf +5 gesetzt wird. Dadurch dauert der Ausdruck vielleicht ein paar Minuten länger, was man jedoch gar nicht bemerkt, da man in der Zeit ungestört seinen Brief weiterschreiben kann.

Ein ähnliches Problem taucht bei Compiler-Sprachen wie z.B. C auf: Versucht man, schon während des Compilier-Vorganges Verbesserungen am Source vorzunehmen, reagiert der ED oft sehr langsam auf die Eingaben. Ein Heruntersetzen der Compiler-Priorität verhindert diesen lästigen Effekt.

9.4 TaskStop - Nichts geht mehr

Wir stellen Ihnen nun ein Hilfsmittel vor, mit dem Sie alle Aktionen in Ihren Amiga jederzeit per Tastendruck für eine beliebige Zeit einfrieren können. Wozu das gut sein soll? Eine Anwendung hierfür werden die Spielefans unter Ihnen sehr begrüßen: Sollte mal wieder im spannendsten Augenblick das Telefon klingeln, genügt ein Druck auf die linke Shift-Taste, und die feindlichen Raumschiffe sind erst einmal auf Eis gelegt. Vielleicht ist ja Ihr Chef am Telefon, der sich erkundigen möchte, wie Sie mit dem neuen Textomatprogramm klarkommen. Sollte Sie das kaltlassen, können Sie anschließend die UFO-Flotte mit Hilfe der rechten Shift-Taste wieder auftauen. Ruft Sie jedoch der Chef zu sich, sollten Sie nun schleunigst den Screen mit der Maus komplett nach unten ziehen und nun diese Situation durch Drücken der linken Shift-Taste einfrieren. Wenn sich die Situation entschärft hat, genügt ein Druck auf die rechte Shift-Taste, um den Amiga wieder zu entriegeln.

Das alles funktioniert natürlich nicht nur bei Spielen. Jedes Programm (notfalls auch das neue Textomat-Programm) läßt sich durch unseren neuen CLI-Befehl unterbrechen. Sie können hierdurch jegliche Einsicht und Manipulation während Ihrer Abwe-

senheit verhindern. Kaum jemand wird auf die Idee kommen, zuerst die rechte Shift-Taste zu betätigen und anschließend mit gedrückter Maus-Taste wieder den versteckten Screen nach oben zu ziehen. Dies alles leistet das nun folgende, relativ kurze C-Programm:

```
#include <exec/types.h>
#include <exec/execbase.h>
#include <exec/tasks.h>
#include <exec/exec.h>
#include <exec/execname.h>
#include <exec/lists.h>
#include <functions.h>

#define SHIFTLINKS 0x3f
#define SHIFTRECHTS 0x3d
#define CONTROL 0x39

extern struct ExecBase *SysBase;

main()
{
struct Task *task;
char *tasten;
tasten = 0xbfec01;  /* Adresse der Sondertasten */
printf ("\n ShiftLinks = Stop, ShiftRechts = Weiter,
                               Control = Ende\n ");
task =  FindTask(0L);
if ( task == 0L)
 {
 printf (" Ich finde meine Task nicht\n");
 exit (0L);
 }
SetTaskPri (task, 127L);
do
 {
 if ( *tasten == SHIFTLINKS )
  {
  do
   {
   i++;
   } while((*tasten != SHIFTRECHTS)&&(*tasten != CONTROL));
  i++;
  }
 Delay(10L);
 } while ( *tasten != CONTROL );
}
```

Zur Funktion des Programms

Nach der üblichen Einbindung der benötigten Headerfiles werden drei Makros definiert. Ihnen werden jeweils bestimmte Tastenwerte zugeordnet. Auch für dieses Programm wird wieder der Zeiger auf die Execbase benötigt. In der Main-Funktion wird zuerst ein Zeiger auf eine Task-Struktur definiert. Der Zeiger *Tasten wird später auf das Hardware-Register gesetzt, aus dem die Tastaturcodes ausgelesen werden können. Dies ist zwar nicht die feine Art, jedoch für unser Programm die einfachste Methode, auf Tastatureingaben schnell reagieren zu können. Aus der merkwürdigen Adresse 0xbfec01 kann ausgelesen werden, ob gerade eine der Sondertasten gedrückt ist. Der Zeiger Tasten wird daher erst einmal mit dieser Adresse gefüttert.

Nach Ausgabe des Benutzungshinweises für das Programm wird durch FindTask der Zeiger auf die eigene Task-Struktur in den Task-Pointer geholt. Sollte der Befehl (aus welchen Gründen auch immer) diese Struktur nicht finden, wird das Programm unter Ausgabe einer entsprechenden Mitteilung vorzeitig beendet: Der Guru bleibt im Hintergrund! Falls die Struktur jedoch gefunden wird, erhält die eigene Task nun die höchste Priorität (+127).

Damit nun andere Tasks trotzdem noch genügend Rechenzeit bekommen, wird in der äußeren Schleife durch den Befehl Delay unser Programmablauf für 10/50 Sekunden verzögert. Während dieses Zeitraums können sich nun andere Programme um den Platz in der Running-Liste streiten. In dieser Zeit wird nun auch nicht ein Druck auf die Shift-Taste ausgewertet, so daß Sie mit einer kleinen Verzögerung bei der Tastaturauswertung rechnen müssen. Ist unsere Task jedoch aktiv und wird nun die linke Shift-Taste betätigt, geht das Programm in eine Schleife, die nunmehr aufgrund unserer hohen Priorität durch keine anderen Tasks mehr unterbrochen werden kann. Lediglich ein Druck auf die rechte Shift-Taste oder auf Ctrl veranlaßt einen Ausstieg aus der Schleife. Bei Ctrl wird gleichzeitig das Programm wieder verlassen und dabei natürlich unser Stop-Task gelöscht.

An dieser Stelle konkrete Beispiele für die Anwendung des TastStop-Befehls zu bringen, ist sicher nicht sehr sinnvoll. Ein

kleiner Tip soll jedoch nicht fehlen: Als Background-Prozeß, die ja bekanntlich auf dem Amiga durch *Run* gestartet werden, funktioniert auch der neue Befehl tadellos. So kann er bei selbstladenden Spieledisketten mit Leichtigkeit in die Startup-Sequence integriert werden.

9.5 Delay - Tasks dynamisch verzögern

Aufgeschoben ist nicht aufgehoben. Das gilt auch für den zuletzt besprochenen TaskStop-Befehl. Denn wenn in einem Spielprogramm die Amiga-Blockade durch die rechte Stop-Taste wieder aufgehoben wird, geht es natürlich im Spiel mit unverminderter Geschwindigkeit weiter. Wir haben uns nun etwas ausgedacht, mit dem wir solche Programme überlisten können: Unser neuer CLI-Befehl Delay ermöglicht es, die Spielgeschwindigkeit jederzeit den eigenen Fähigkeiten anzupassen. Der große Vorteil liegt darin, daß zwar das eigentliche Programm langsamer wird, nicht aber die Reaktionen auf unsere Eingaben.

Die beiden Shift-Tasten haben bei diesem Programm eine nicht ganz so krasse Funktion wie bei dem TaskStop-Befehl: Mit ihnen lassen sich Programme dynamisch verzögern, das heißt, je länger eine Taste betätigt wird, desto stärker ist die Auswirkung. Zusätzlich haben wir ein weiteres Feature in das Programm eingebaut. Sie möchten doch sicherlich, daß sich alle Programme mit dem neuen Befehl verzögern lassen. Also muß der Befehl gegebenenfalls eine hohe Priorität haben, wenn das Spieleprogramm beispielsweise auch seine Priorität erhöht hat. Andererseits muß der Befehl eine möglichst niedrige Priorität haben, damit die System-Tasks durch ihn nicht behindert werden. Ist die Priorität beispielsweise höher als 20, so gilt die Verzögerung auch für die Maus. Diese wird dann nur noch ruckartig vorwärtsbewegt - für ein schönes Spiel eine echte Zumutung. Alle diese Probleme löst der Befehl Delay mit links. Aber: erst die Arbeit ...

Das C-Listing von Delay

```
/* Control = Ende, Shiftlinks = langsamer, Shiftrechts = schneller */

#include <exec/types.h>
#include <exec/tasks.h>
#include <functions.h>
#define CONTROL 0x39
#define SHIFTLINKS 0x3f
#define SHIFTRECHTS 0x3d
extern struct ExecBase *SysBase;
main(argc, argv)
int argc;
char *argv[];
   {
   char *taste;
   struct Task *task;
   long Wert;
   long pri;
   int ergebnis;
   long i;
   Wert = 1L;
   taste = 0xbfec01;
   pri = 0L;                    /* Priorität mit 1 vorgeben */
   if ((argc != 2) && (argc != 3))
      {
      printf(" Verzögerung [Priorität] \n");
      exit(FALSE);
      }
   ergebnis = Value(argv[1], &Wert);
   if(ergebnis == FALSE)
      {
      printf (" 0 < Wartezeit < 999, nicht: %ld\n", Wert);
      exit(FALSE);
      }
   if (argc == 3)
      {
      ergebnis = Value(argv[2], &pri);

      if( (ergebnis == FALSE) || (pri < -127) || (pri > 127) )
         {
         printf (" -127 < Priorität < 127, nicht: %ld\n", pri);
         exit(FALSE);
         }
      }
   task = FindTask(0L);
   if (task == 0L)
   {
      printf ("Finde meine Task nicht!\n");
      exit(0L);
   }
   SetTaskPri (task, pri);

       do
```

```
      {
        i = 0L;
        Delay( 5L );
        for ( i = 0L; i < Wert * 10L; i++ )
           {
           while ( (*taste == SHIFTLINKS ) && (Wert < 995L ) )
              {
              Wert++;
              Wert++;
              Wert++;
              Wert++;
              Wert++;
              Delay(1L);
              }
           while ( (*taste == SHIFTRECHTS ) && (Wert > 5L ) )
              {
              Wert--;
              Wert--;
              Wert--;
              Wert--;
              Wert--;
              Delay(1L);
              }
           }
      } while (*taste != CONTROL);
      printf (" Wert = %ld\n", Wert);
}


/* Aus bis zu drei Ziffern Zahl 0 - 999 machen */

Value(string, wert)
char *string;
long *wert;
   {
   int i;
   int anzziffern;
   char zeichen;
   int ziffer;
   int zehner;
   zehner = 1;
   anzziffern = 0;
   *wert = 0L;
   if (*string == '-')      /* negative Zahl */
      {
      zehner = -1;          /* in negative Zahl wandeln */
      string++;             /* Minuszeichen überlesen */
      }
      for (i = 1; i <= 4; i++)     /* Anzahl Ziffern bestimmen */
      {
      if (*string == 0)
         break;
      string++;
      anzziffern++;
```

```
    }
  if ((anzziffern == 0) || (anzziffern > 3)) /* nur drei Ziffern
                                                erlaubt */
    {
    printf (" Zuviele Ziffern %d\n", anzziffern);
    return (FALSE);
    }
  for (i = anzziffern; i > 0; i-- )  /* Zahl bilden */
    {
    string--;
    zeichen = *string;              /* String jetzt von hinten lesen*/
    if (zeichen < '0' || zeichen > '9')
      {
      printf ("Falsche Ziffer %c \n", zeichen );
      return (FALSE);
      }
    ziffer = zeichen - '0';
    *wert = *wert + ziffer * zehner;
    zehner = zehner * 10;
    }
/*  if (*wert <= 999 || *wert > 999)
    return (FALSE);                                    */
  return (TRUE);
  }
```

Zur Funktion des Programms

Der Programmkopf ist direkt von TaskStop übernommen. Bei der Auswertung der Eingabezeile werden folgende drei Fälle unterschieden:

1. Es sind nicht zwei oder drei Parameter übergeben worden. In diesem Fall wird das Programm mit der Ausgabe des korrekten Aufrufs abgebrochen.

2. Es sind genau zwei Parameter übergeben worden. Dann wird die Verzögerung in eine Zahl umgerechnet und die vorgegebene Priorität Null nicht geändert.

3. Es sind genau drei Parameter übergeben worden. Dann wird auch der zweite Parameter in einen Wert umgerechnet, auf gültigen Wertebereich getestet und der Variablen Pri zugeordnet.

Anschließend wird mit FindTask ein Zeiger auf die eigene Task-Struktur ermittelt und mit SetTaskPri die neue Priorität festgelegt. War keine Priorität angegeben worden, so hat unsere Task unabhängig vom aufrufenden CLI-Prozeß immer die Prio-

rität Null. Die eigentliche Arbeit geschieht in der Do-While-Schleife. Diese besteht aus drei Komponenten:

1. Einem Aufruf der Wartefunktion Delay. Die stellt sicher, daß auch bei hohen Prioritäten die System-Task genügend Zeit bekommt.

2. Einer Verzögerungsschleife, deren Dauer abhängig vom eingestellten Verzögerungswert ist. Während dieser Zeit werden alle Tasks blockiert, die kleinere Priorität haben.

3. Einem doppelten Abfrageblock, in dem die beiden Shift-Tasten kontrolliert werden. Wird eine der Tasten gedrückt, so prüft das Programm, ob der Verzögerungswert noch im erlaubten Bereich ist. Ist dies der Fall, so wird der Verzögerungswert entsprechend verändert. Einiges Kopfzerbrechen werden Ihnen wohl die fünffach vorhandenen Zeilen Wert++ bzw. Wert-- und das anschließende Delay(1L) bereiten. Das Delay verhindert, daß bei einem Druck auf eine Shift-Taste der Wert sofort Maximum oder Minimum erreicht. Die fünffach vorhandenen Zeilen haben eine doppelte Aufgabe: Einerseits sorgen sie dafür, daß sich der Wert schnell genug ändern läßt, denn eine kleinere Verzögerung als 1L ist nicht möglich. Andererseits verhindern sie, daß das gebremste Programm beim Druck auf eine Shift-Taste ungehindert weiterläuft.

   Würde der Abfrageblock ausschließlich aus einem Delay bestehen, so würde das Programm praktisch überhaupt nicht mehr gebremst. Da hier aber ein gewisser Programmcode immer mit der hohen Priorität abgearbeitet wird, wird das zu verzögernde Programm zumindest etwas abgebremst. Am besten probieren Sie das einmal aus, indem Sie ein Programm mit einem Verzögerungswert 500 abbremsen. Sie werden dann feststellen, daß bei jedem Druck auf eine Shift-Taste das Programm deutlich beschleunigt wird.

9.6 Beliebige Zeichenketten in Dateien ersetzen - Ersetze

Sicherlich haben Sie unter den vorhandenen CLI-Befehlen schon einen Befehl vermißt, der in Dateien Zeichen oder Zeichenket-

ten durch andere ersetzen kann. Jede bessere Textverarbeitung kennt eine solche Funktion. Zwar gibt es den Befehl Search, um Zeichenketten in Dateien zu suchen, aber durch eine andere Zeichenfolge ersetzen kann man mit diesem Befehl nichts. Wir haben deshalb für Sie einen neuen CLI-Befehl entwickelt, der diese Möglichkeit bietet und dabei schnell und leistungsfähig ist: Ersetze.

Von großem Vorteil bei diesem Befehl ist, daß das Ergebnis stets in einer neuen Datei steht. Sollten Sie also einmal einen Fehler gemacht haben, haben Sie immer noch die ursprüngliche Datei unverändert. Zusätzlich erlaubt es der Befehl, beliebige Zeichen zu suchen und durch beliebige Zeichen zu ersetzen. Das bietet kaum eine Textverarbeitung. So können Sie beispielsweise Carrige Return durch Leerzeichen ersetzen oder jedes Leerzeichen durch 100 Punkte. Um beliebige Zeichen eingeben zu können, erfolgt die Eingabe durch die zugehörigen ASCII-Zahlen. Ein Leerzeichen ist also eine 32 und ein Carriage Return eine 13. Da es aber sehr umständlich ist, Zeichenketten als Zahlen einzugeben, können Sie wahlweise auch einfach die Zeichenketten eingeben. Ersetze prüft jeweils das erste Zeichen der Eingabe. Ist es eine Ziffer, so nimmt es die folgenden Zeichen als ASCII-Werte, ansonsten als Zeichenkette.

Kommen wir nun aber zum Aufruf des neuen CLI-Befehls. Er lautet:

```
Filenamealt Filenameneu String
```

Filenamealt ist dabei der Name der Datei, aus der gelesen werden soll. Hierbei kann der komplette Pfad angegeben werden.

Filenameneu ist der Name der neuen Datei, in die geschrieben werden soll.

Achtung: Ersetze prüft nicht, ob die Datei besteht, und überschreibt bestehende Dateien, wie es auch alle anderen CLI-Befehle machen.

String ist die Angabe, wonach gesucht und durch was ersetzt werden soll. Dabei werden Such- und Ersetze-String durch einen

Doppelpunkt getrennt und dürfen (CLI-typisch) keine Leerzeichen enthalten. Ein mögliches Beispiel für einen Aufruf wäre also:

```
Ersetze AlteDatei NeuDatei Meier:Mayer
```

Dabei wäre in der anschließend erzeugten Datei NeuDatei immer Meier durch Mayer ersetzt. In diesem Fall wurden also Zeichenketten eingegeben. Ersetze erkennt das daran, daß weder Meier noch Mayer mit einer Ziffer beginnen (0 - 9). Nehmen wir nun ein weiteres Beispiel, bei dem die Eingabe als ASCII-Werte erfolgt. Normalerweise beendet ein RETURN die Eingabe, folglich kann kein Carriage Return (CR) eingegeben werden. Nicht so bei unserem neuen Befehl. Der Aufruf

```
Ersetze AlteDatei NeuDatei 13:32
```

ersetzt jedes CR (13) durch ein Leerzeichen (32). Zusätzlich ist auch die Kombination beider Eingabemöglichkeiten erlaubt. So ersetzt beispielsweise

```
Ersetze AlteDatei NeuDatei 32:.....
```

jedes Leerzeichen durch 5 Punkte. Wollen Sie einen String aus seinen ASCII-Werten zusammensetzen, so trennen Sie die einzelnen Zahlen bitte durch ein Komma. So ersetzt beispielsweise

```
Ersetze AlteDatei NeuDatei 32,32,32:32
```

jeweils 3 Leerzeichen durch ein einziges.

Die Länge des Such-Strings beziehungsweise Ersetze-Strings ist auf 127 Zeichen begrenzt, was für den normalen Gebrauch wohl völlig ausreichen dürfte. Benötigen Sie also beispielsweise eine Testdatei aus beliebigen Zeichen, so können Sie diese folgendermaßen erzeugen:

Erstellen Sie zuerst eine Ausgangsdatei mit einigen gleichen Zeichen durch:

```
ECHO >RAM:TEST "aaaaaaaaaaaaaaaaaaaa"
```

Nun ersetzen Sie einfach jedes a durch sehr viele neue a mit:

```
Ersetze RAM:TEST RAM:TEST1 a:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Nun ist Ihre Datei schon erheblich länger geworden. Wenn Ihnen das nicht reicht, rufen Sie Ersetze eben erneut auf mit:

```
Ersetze RAM:TEST1 RAM:TEST2 a:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Wenn Sie dies Spielchen noch einige Male weitertreiben, heißt es "Volume RAM is full" und selbst eine Festplatte dürfte nur noch ein weiteres Mal ausreichen.

Wenn Sie eine Zeichenkette ersatzlos streichen wollen, müsse Sie hinter dem Doppelpunkt nur die Eingabe weglassen. So löscht beispielsweise

```
Ersetze AlteDatei NeueDatei 32:
```

jedes Leerzeichen aus der Datei. Ein anschließendes

```
Type NeueDatei
```

beweist das schnell. Denken Sie aber bitte daran, daß jeweils das erste Zeichen darüber entscheidet, ob die Eingabe als Zeichenkette oder als Zahl ausgewertet wird. So ist die Eingabe

```
Ersetze AlteDatei NeueDatei 3Tage:4Stunden
```

nicht erlaubt. In diesem Fall möchten wir darauf verzichten Ihnen die genaue Funktionsweise des Programms zu erklären, denn diese hat kaum noch etwas mit AmigaDOS zu tun. Außerdem ist es für alle Programmierer wohl ausreichend strukturiert und kommentiert. Ein kleiner Hinweis noch für alle interessierten Programmierer: Unser Programm benutzt einen Ringpuffer zur Ablage der schon gelesenen, aber nicht ausgewerteten Zeichen. In diesem Ringpuffer wird solange gepuffert, bis entschieden werden kann, ob der Such-String gefunden wurde oder nicht. Einen solchen Ringpuffer kann man mit dem Modulo-Befehl in C realisieren. Dann hätte aber bei jedem Weitersetzen eines Zeigers, der im Puffer arbeitet, der MOD-Befehl eingesetzt werden müssen, der doch erhebliche Zeit in Anspruch nimmt. Statt dessen haben wir eine scheinbar umständlichere, aber viel schnellere Methode gewählt. Das Unterprogramm zum Einlesen neuer Zeichen prüft, ob das Ende des Puffers erreicht wurde (Naht)

und verschiebt in einem solchen Fall die gültige Eingabe an den Anfang des Puffers. So können alle anderen Routinen mit ihren Zeigern ohne jede Prüfung oder MODULO-Bildung auf den Puffer zugreifen.

```
/* Dies Programm ersetzt beliebige Zeichenketten in Dateien */

#include <stdio.h>

#define FALSCH 0                    /* Konstanten, erleichtern das Lesen */
#define WAHR 1                      /* des Programmtextes                */
#define byte unsigned char
#define Maxpuffer 512               /* Größe des Lesepuffers */

 void Verschiebe();
 char Vonstring[127];               /* Such-String */
 char Nachstring[127];              /* Ersetzungs-String */
 char Pufferein[Maxpuffer];         /* Lesepuffer */
 int Pschreib;                      /* Stelle im Puffer für Wegschreiben */
 int Plies;                         /* Stelle im Puffer für Einlesen */

 unsigned long Anz;                 /* Zähler für Zeichen */
 unsigned int Vonlen;
 unsigned int Nachlen;
 unsigned int Anzersetzt;           /* Zähler für : wie oft ersetzt */
 FILE *Rein, *Raus;



/* -------------------- Parameter holen ------------------------- */
void GetPara(Para, Von, Nach)
char *Para;
char *Von;
char *Nach;
{
 int Zahl;
 int Laenge;       /* Zähler für Länge, Abschluß 0 wird nicht gezählt */

 *Von = 0;
 *Nach = 0;
 Laenge = 0;             /* Länge auf 0 setzen */

 Zahl = atoi(Para);      /* Prüfen, ob String oder ASCII-Werte */
 if (Zahl != 0)          /* ASCII-Werte für String1 */
 {
   while ((*Para != 0)&&(*Para != ':'))
                         /* Solange noch Parameter für String1 */
   {
      Zahl = atoi(Para);
      if (Zahl > 255)          /* nicht erlaubte Zahl */
      {
         printf("Zahl ist kein ASCII-Wert\n");
```

```
         exit();
      }
      *(Von++) = (char) Zahl;      /* Zeichen ablegen */
      Laenge++;
      while ((*Para != 0)&&(*Para != ',')&&(*Para != ':'))
         Para++;                   /* Trennung suchen */
      if (*Para == ',') Para++;    /* überlesen */
   }                               /* Ende While */
   *Von = 0;                       /* String1 mit 0 abschließen */
   Vonlen = Laenge;                /* Länge von String1 merken */
   }                               /* Ende if( ASCII) */

else                               /* String1 ist ein String */
{
   for(Laenge = 0; (*Para!=0)&&(*Para!=':'); Para++)
   {
      *(Von++) = *Para;
      Laenge++;
   }
   *Von = 0;                       /* Such-String mit 0 abschließen */
   Vonlen = Laenge;                /* Länge des Such-Strings merken */
}                                  /* Ende else */

if (Vonlen == 0)
   {
      printf("Such-String muß angegeben werden!\n");
      exit();
   }


if (*Para != ':')
   {
      printf("Doppelpunkt fehlt!\n");
      exit();
   }

Para++;                            /* Para hinter Doppelpunkt setzen */
Laenge = 0;                        /* Länge auf 0 setzen */
Zahl = atoi(Para);                 /* Prüfen, ob String oder ASCII-Werte */
if (Zahl != 0)                     /* ASCII-Werte für String2 */
{
   while (*Para != 0)              /* Solange noch Parameter für String2 */
   {
      Zahl = atoi(Para);
      if (Zahl > 255)              /* nicht erlaubte Zahl */
      {
         printf("Zahl ist kein ASCII-Wert\n");
         exit();
      }
      *(Nach++) = (char) Zahl;     /* Zeichen ablegen */
      Laenge++;                    /* Länge erhöhen */
      while ((*Para != 0)&&(*Para != ',')&&(*Para != ':'))
         Para++;                   /* Trennung suchen */
      if (*Para == ',') Para++;    /* überlesen */
```

```
    }                                   /* Ende While */
 *Nach = 0;                             /* String2 mit 0 abschließen */
 Nachlen = Laenge;                      /* Länge von String2 merken */
 }                                      /* Ende if(ASCII)
 else                                   /* String2 ist ein String */
 {
    for(Laenge = 0; (*Para!=0); Para++)
    {
       *(Nach++) = *Para;
       Laenge++;
    }
    *Nach = 0;                      /* Ersetze-String mit 0 abschließen */
    Nachlen = Laenge;               /* Länge des Such-Strings merken */
 }                                  /* Ende else */

}

int Lies(Anzahl)
int Anzahl;
{
 register char Zeichen;
 int gelesen;

 gelesen = 0;

 for(gelesen = 0; gelesen < Anzahl; gelesen++)
 {
   Zeichen = fgetc(Rein);
   if (Zeichen == EOF)
     return(gelesen);                 /* Kein Zeichen mehr in Datei */
   Anz++;                             /* Anzahl Zeichen erhöhen */
   Pufferein[Plies++] = Zeichen;
   if (Plies >= Maxpuffer)          /* Puffer zu Ende, über die Naht */
      Verschiebe();                 /* Inhalt bündig an Anfang schieben */
 }
 return(gelesen);
}

void Verschiebe()           /* Inhalt im Puffer über Naht verschieben */
 {
  int wieviel;
  int i;

  wieviel = Plies - Pschreib;
  for (i = 0; i< wieviel; i++)
     Pufferein[i] = Pufferein[Pschreib+i];
  Pschreib = 0;             /* Indizes neu setzen */
  Plies = i;
 }

int Checkpuffer()           /* Vergleiche Such-String mit Puffer */
{
 char *string;
```

```
 int i;

 i = Pschreib;
 for (string = &Vonstring[0]; *string != 0; string++)
 {
    if (*string != Pufferein[i])
       return(FALSCH);
    i++;
 }
 return(WAHR);
}

void Schreibzeichen()    /* Ein Zeichen aus Puffer in Datei schreiben */
{
 register char Zeichen;

 Zeichen = Pufferein[Pschreib];
 fputc(Zeichen, Raus);
 Pschreib++;             /* Schreibzeiger weitersetzen */
}

void Ersetze()           /* Ersetze-String schreiben, Puffer weg */
{
 int i;
 char *string;
 register char Zeichen;

 i = 1;
 for (string = &Nachstring[0]; *string != 0; string++)
 {
    Zeichen = *string;
    fputc(Zeichen, Raus);
    i++;
 }
 Pschreib = Plies;    /* Schreib- auf Lesezeiger -> Puffer leer */
}

void Schreibrest()    /* Keine Zeichen mehr in Datei, schreibe Puffer */
{
 register char Zeichen;
 int i;

i = 0;
 for(i = Pschreib; i < Plies; i++)
 {
    Zeichen = Pufferein[i];
    putc(Zeichen, Raus);
 }
}



main (Anzahl, Argumente)
```

```
int Anzahl;
char *Argumente[];
{
 int Anzlies;
 int Anzgelesen;
 int Error;
 if (Anzahl != 4)
    {
        printf("Dieses Programm ersetzt beliebige Zeichenketten \n");
        printf("Copyright Manfred Tornsdorf\n");
        printf("Aufruf: Filenamealt Filenameneu String\n");
        printf("entweder :String = Zahl,Zahl,Zahl...:
                                   Zahl,Zahl,Zahl...\n");
        printf("oder     :String = string:string\n\n");
        printf("Beispiel :Ersetze t.txt tt.txt Meintext:32,33,32\n");
        printf("ergibt   :<Meintext> -> < ! >\n");
        printf("Fehlt Ersetzungs-String, so wird durch NICHTS
                                        ersetzt/gelöscht\n");
    exit();
        }

 GetPara(Argumente[3], &Vonstring, &Nachstring );
              /* Such- und Ersetz-Strings erstellen */
 Vonlen = strlen(Vonstring);
 Nachlen = strlen(Nachstring);

 Rein = fopen(Argumente[1], "r");
 Raus = fopen(Argumente[2], "w");
 if (Rein == 0)
 {
 puts ("Ich kann die Eingabedatei nicht öffnen");
 exit ();
 }
 if (Raus == 0)
 {
 puts ("Ich kann die Ausgabedatei nicht öffnen");
 exit ();
 }
 Anz = 0;
 Pschreib = 0;          /* Schreibzeiger auf Pufferanfang */
 Plies = 0;             /* Lesezeiger auf Pufferanfang */
 Anzlies = Vonlen;      /* Such-String-Zeichen lesen */
 Anzersetzt = 0;        /* Default: nicht gefunden */

 Anzgelesen = 1;        /* Default, um TC zu beruhigen */
 while(Anzgelesen != 0)
 {
    Anzgelesen = Lies(Anzlies); /* notwendige Anzahl aus Datei lesen */
    Error = Checkpuffer();
    if(Error == FALSCH)     /* nicht gefunden */
    {
       Schreibzeichen();    /* 1. Zeichen schreiben */
       Anzlies = 1;         /* nur ein Zeichen lesen */
    }
```

```
    else                    /* gefunden */
    {
      Ersetze();            /* Ersetz-String schreiben, Puffer weg */
      Anzersetzt++;         /* Zähler für Ersetzt erhöhen */
      Anzlies = Vonlen;
    }
  }   /* Ende while(), Datei leer */
  Schreibrest();            /* letzte Zeichen aus Puffer schreiben */

  printf("\nZeichenkette %d mal ersetzt\n", Anzersetzt);
  fclose(Rein);
  fclose(Raus);
  exit();


}
```

10. Quick-Reference für ED, Edit und CLI-Kommandos

Die folgenden zwei Unterkapitel geben einen geordneten Überblick über die Tastenkombinationen und deren Auswirkungen bei den Editoren ED und Edit. Diese werden in einer Tabellenform aufgelistet und sollen Ihnen als Nachschlagewerk dienen.

Im dritten Unterkapitel werden die CLI-Kommandos in einer ähnlichen Form dargelegt. Durch die größere Übersichtlichkeit hoffen wir Ihnen ein schnelleres Arbeiten zu ermöglichen.

10.1 Die ED-Funktionen

Es gibt beim ED-Editor zwei verschiedene Arten von Befehlen. Zum einen gibt es die Befehle, die direkt ausgeführt werden, nachdem die Tastenkombination gedrückt wurde, und zum andern gibt es die Befehle, die erst im Kommandomodus eingegeben werden können.

Zunächst möchten wir einiges über die sogenannten direkten Befehle schreiben, die immer aus einer Tastenkombination bestehen. Diese Tastenkombination beginnt immer mit der Ctrl-Taste. Gleichzeitig dazu muß eine weitere Taste gedrückt werden. Beide zusammen haben dann eine sofortige Auswirkung auf den Text oder auf den Cursor.

Damit die Liste der direkten Befehle und der Kommandomodus-Befehle nicht durch einen Text getrennt werden, folgen nun erst noch einige Zeilen zum Kommandomodus.

In den Kommandomodus gelangen Sie, indem Sie die ESC-Taste drücken. Ob Sie im Kommandomodus sind, erkennen Sie daran, daß in der linken unteren Ecke des Editorfensters ein Sternchen zu sehen ist. Sie müssen dann nur noch den Befehl eingeben und die Return-Taste betätigen. Erst danach hat der eingegebene Befehl auch Auswirkungen auf Text oder Cursor.

Da es sowohl in der einen als auch in der anderen Gruppe Befehle gibt, die die gleichen Funktionen erfüllen, müssen Sie sich entscheiden, mit welchem Befehlstyp Sie besser arbeiten können.

Es folgt jetzt eine Liste von Befehlen, die man auch zu den direkten Befehlen zählen kann, da sie sich auch unmittelbar auf den Text oder den Cursor auswirken. Sie unterscheiden sich allerdings darin, daß die Funktionen nicht in Kombination mit der Ctrl-Taste aktiviert werden. Daran anschließend folgen die Listen der direkten und Kommandomodus-Befehle. Sie sind geordnet nach Funktionen.

Direkte Befehle (ohne Ctrl)

Tab	Bewegt den Cursor zur nächsten Tabulatormarke.
Del	Löscht das Zeichen unter dem Cursor.
Backspace	Löscht das Zeichen links neben dem Cursor.
Return	Text zwischen Cursor-Position und Zeilenende wird in nächste Zeile übernommen.

Direkte Befehle (mit Ctrl)

Ctrl und I	Bewegt den Cursor zur nächsten Tabulatormarke.
Ctrl und U	Bewegt den Cursor 12 Zeilen nach oben.
Ctrl und D	Bewegt den Cursor 12 Zeilen nach unten.
Ctrl und R	Bewegt Cursor auf das Ende des links von ihm liegenden Wortes.
Ctrl und T	Bewegt den Cursor auf den Anfang des nächsten Wortes.
Ctrl und]	Bewegt den Cursor auf den Zeilenanfang bzw. das Zeilenende.
Ctrl und E	Bewegt den Cursor zum Fensteranfang oder Fensterende.
Ctrl und H	Löscht das Zeichen links neben dem Cursor.
Ctrl und O	Löscht ein Wort oder alle Leerstellen bis zum nächsten Wort.
Ctrl und Y	Löscht alles von Cursor-Position bis Zeilenende.
Ctrl und B	Löscht die gesamte aktuelle Zeile.
Ctrl und M	Text zwischen Cursor-Position und Zeilenende wird in nächste Zeile übernommen.
Ctrl und A	Fügt eine Zeile ein.
Ctrl und G	Der letzte Kommandomodus-Befehl wird wiederholt.
Ctrl und [	Es wird in den Kommandomodus gesprungen.
Ctrl und F	Zeichen unter dem Cursor wechselt von Groß- auf Kleinschrift oder umgekehrt.

Kommandomodus-Befehle (mit ESC)

M n	Bewegt den Cursor in die n-te Zeile.
CL	Bewegt den Cursor eine Position nach links.
CR	Bewegt den Cursor eine Position nach rechts.
CS	Bewegt den Cursor an den Zeilenanfang.
CE	Bewegt den Cursor an das Zeilenende.
P	Bewegt den Cursor an den Anfang der vorigen Zeile.
N	Bewegt den Cursor an den Anfang der nächsten Zeile.
T	Bewegt den Cursor an den Textanfang.
B	Bewegt den Cursor an das Textende.
BS	Markiert Cursor-Position als Blockanfang.
BE	Markiert Cursor-Position als Blockende.
SB	Zeigt den markierten Block im Fenster.
WB "Datei"	Markierter Block wird in Datei abgespeichert.
IB	Markierter Block wird an die Cursor-Position kopiert.
DB	Markierter Block wird gelöscht.
DC	Zeichen unter dem Cursor wird gelöscht.
D	Gesamte aktuelle Zeile wird gelöscht.
S	Text zwischen Cursor-Position und Zeilenende wird in nächste Zeile übernommen.
J	Verbindet die aktuelle Zeile mit der nächsten.
I "Text"	Text wird vor der aktuellen Zeile eingefügt.
A "Text"	Text wird nach der aktuellen Zeile eingefügt.
IF Datei	Datei wird an der Cursor-Position eingefügt.
E "Text1"Text2"	Text1 wird gegen Text2 ausgetauscht.
EQ "Text1"Text2"	Text1 wird mit vorheriger Abfrage gegen Text2 ausgetauscht.
F "Text"	Sucht ab Cursor-Position nach Text.
BF "Text"	Sucht bis Cursor-Position nach Text.
LC	Bei Textsuche wird Unterscheidung zwischen Groß- und Kleinschrift eingeschaltet.
UC	Bei Textsuche wird Unterscheidung zwischen Groß- und Kleinschrift ausgeschaltet.
X	ED wird beendet, und der Text wird abgespeichert.
Q	ED wird beendet, und der Text wird nicht abgespeichert.
SA	ED wird nicht beendet, und der Text wird abgespeichert.
RP Befehl	Befehl wird so oft wiederholt, bis ein Fehler auftritt.
SH	Aktuelle Einstellungen des Editors werden angezeigt.
U	Änderungen an der aktuellen Zeile werden rückgängig gemacht.
SL n	Linker Rand wird auf n festgelegt.
SR n	Rechter Rand wird auf n festgelegt.
EX	In der aktuellen Cursor-Zeile wird der rechte Rand aufgehoben.

10.2 Die Edit-Funktionen

Außer dem im vorigen Kapitel besprochenen ED-Editor ist auf der Workbench-Diskette noch ein weiterer Editor vorhanden. Es ist der Edit-Editor. Wir möchten Ihnen hier einen kurzen Überblick über die möglichen Edit-Befehle geben. Die gleich folgende Liste der Befehle soll allerdings nur als Nachschlagewerk dienen, nicht aber als ausführliche Anleitung. Nähere Informationen zur Handhabung des Editors erhalten Sie im Kapitel 2.

Bevor wir nun die Befehle auflisten, noch einiges vorweg. Zusammen mit den Befehlsworten werden teilweise auch noch Argumente übergeben. Dabei dient der Slash / als Trennzeichen für Zeichenketten. Argumente, die wahlweise angegeben werden können, sind in Klammern () gefaßt. Damit der Befehlstext nicht zu lang wird, haben wir folgende Abkürzungen benutzt:

```
a,b = Zeilennummern (oder . oder *)
bg = Befehlsgruppe
m,n = Zahlen
q = Qualifizierer
sk = Suchkriterium
sw = Schaltwert (+ oder -)
z1,z2 = Zeichenketten
```

Doch nun zu den Befehlen:

<

Der Zeichenzeiger wird um ein Zeichen nach links versetzt.

>

Der Zeichenzeiger wird um ein Zeichen nach rechts versetzt.

#

Das Zeichen an der Zeichenzeigerposition wird gelöscht.

$

Das Zeichen an der Zeichenzeigerposition wird in Kleinbuchstaben umgewandelt.

%

Das Zeichen an der Zeichenzeigerposition wird in Großbuchstaben umgewandelt.

-

Das Zeichen an der Zeichenzeigerposition wird in eine Leerstelle umgewandelt.

PA (q)/z1/

Der Zeichenzeiger wird auf die Position nach der gegebenenfalls qualifizierten Zeichenkette z1 gesetzt.

PB (q)/z1/

Der Zeichenzeiger wird auf die Position vor der gegebenenfalls qualifizierten Zeichenkette z1 gesetzt.

PR

Der Zeichenzeiger wird auf den Zeilenanfang zurückgesetzt.

M n

Es wird auf Zeile n positioniert.

M +

Es wird auf die zuletzt aus der Quelldatei gelesene Zeile positoniert.

M -

Es wird auf die zuerst im Puffer vorhandene Zeile positioniert.

N

Es wird auf die nächste Zeile positioniert.

P

Es wird auf die vorhergehende Zeile positioniert.

Rewind

Die Quelldatei wird "zurückgespult".

F ((q)/sk/)

Es wird in Vorwärtsrichtung nach der Zeile gesucht, die die gegebenenfalls qualifizierte Zeichenkette sk enthält.

BF ((q)/sk/)

Es wird in Rückwärtsrichtung nach der Zeile gesucht, die die gegebenenfalls qualifizierte Zeichenkette sk enthält.

DF ((q)/sk/)

Es wird in Vorwärtsrichtung nach der Zeile gesucht, die die gegebenenfalls qualifizierte Zeichenkette sk enthält. Dabei werden alle überlesenen Zeilen gelöscht.

?

Die aktuelle Zeile wird verifiziert.

!

Die aktuelle Zeile wird mit allen nicht druckbaren Zeichen verifiziert.

T

Die Quelldatei wird bis zum Dateiende angezeigt.

T n

Die nächsten n Zeilen der Quelldatei werden angezeigt.

TL n

Die nächsten n Zeilen der Quelldatei werden mit Zeilennummern angezeigt.

TN

So viele Zeilen werden angezeigt, wie in den Textpuffer passen.

TP

Es werden alle Zeilen des Textpuffers angezeigt, und es wird auf den Textpufferanfang positioniert.

V sw

Es wird die Verifizierung ein- (+) oder ausgeschaltet (-).

A (q)/z1/z2/

Eine Zeichenkette z2 wird nach der gegebenenfalls qualifizierten Zeichenkette z1 eingefügt.

AP (q)/z1/z2/

Eine Zeichenkette z2 wird nach der gegebenenfalls qualifizierten Zeichenkette z1 eingefügt. Anschließend wird der Zeichenzeiger hinter z2 positioniert.

B (q)/z1/z2/

Eine Zeichenkette z2 wird vor der gegebenenfalls qualifizierten Zeichenkette z1 eingefügt.

BP (q)/z1/z2/

Eine Zeichenkette z2 wird vor der gegebenenfalls qualifizierten Zeichenkette z1 eingefügt. Anschließend wird der Zeichenzeiger hinter z1 positioniert.

CL (/z1/)

Es werden die aktuelle Zeile, z1 und die nächste Zeile zu einer neuen Zeile zusammengefaßt.

D

Die aktuelle Zeile wird gelöscht.

DFA (q)/z1/

Es wird nach der gegebenenfalls qualifizierten Zeichenkette z1 der Rest der aktuellen Zeile gelöscht.

DFB (q)/z1/

Es wird vom Anfang der gegebenenfalls qualifizierten Zeichenkette z1 der Rest der aktuellen Zeile gelöscht.

DTA (q)/z1/

Es wird vom Anfang der aktuellen Zeile bis einschließlich der gegebenenfalls qualifizierten Zeichenkette z1 gelöscht.

DTB (q)/z1/

Es wird vom Anfang der aktuellen Zeile bis ausschließlich der gegebenenfalls qualifizierten Zeichenkette z1 gelöscht.

E (q)/z1/z2/

Die gegebenenfalls qualifizierte Zeichenkette z1 wird durch die Zeichenkette z2 ersetzt.

EP

Die gegebenenfalls qualifizierte Zeichenkette z1 wird durch die Zeichenkette z2 ersetzt. Anschließend wird der Zeichenzeiger hinter z2 positioniert.

I (a)

Es wird vor der aktuellen Zeile oder der Zeile a Text von der Tastatur eingefügt, bis z gegeben wird.

I z1

Es wird vor der aktuellen Zeile der gesamte Inhalt der Datei z1 eingefügt.

R (a(b))

Die Zeilen a bis b werden gelöscht und anschließend wird Text von der Tastatur eingefügt.

R (a(b)) z1

Die Zeilen a bis b werden gelöscht und anschließend wird der Inhalt der Datei z1 eingefügt.

SA (q)/z1/

Die aktuelle Zeile wird nach dem ersten Auftreten der gegebenenfalls qualifizierten Zeichenkette z1 geteilt.

SB (q)/z1/

Die aktuelle Zeile wird vor dem ersten Auftreten der gegebenenfalls qualifizierten Zeichenkette z1 geteilt.

GA (q)/z1/z2/

Es wird bei allen aktuellen Zeilen die Zeichenkette z2 nach der gegebenenfalls qualifizierten Zeichenkette z1 eingefügt.

GB (q)/z1/z2/

Es wird bei allen aktuellen Zeilen die Zeichenkette z2 vor der gegebenenfalls qualifizierten Zeichenkette z1 eingefügt.

GE (q)/z1/z2/

Es wird die gegebenenfalls qualifizierte Zeichenkette z1 in allen aktuellen Zeilen durch die Zeichenkette z2 ersetzt.

CG (n)

Die globale Operation n oder alle globalen Operationen werden abgeschaltet.

DG (n)

Die globale Operation n oder alle globalen Operationen werden zeitweilig abgeschaltet.

EG (n)

Es werden die zeitweilig abgeschalteten globalen Operationen wieder eingeschaltet.

SHG

Es werden Informationen über alle bisher aktiven globalen Operationen angezeigt.

From

Die ursprüngliche From-Datei wird zur aktuellen Quelldatei.

From .z1.

Die Datei z1 wird zur aktuellen Quelldatei.

To

Die ursprüngliche To-Datei wird zur aktuellen Zieldatei.

To .z1.

Die Datei z1 wird zur aktuellen Zieldatei.

CF .z1.

Es wird die Datei z1 geschlossen.

'

Der zuletzt eingegebene A-, B- oder E-Befehl wird wiederholt.

= n

Der aktuellen Zeile wird die Nummer n zugewiesen.

C .z1.

Es werden alle weiteren Edit-Befehle aus der Datei z1 gelesen.

H n

Der nächste Haltepunkt wird bei Zeile n gesetzt (Bei n=*, anhalten und Haltepunkt löschen.)

Q

Die Befehlsebene wird verlassen. Wird die höchste Befehlsebene verlassen, wird der Rest der Quelldatei übertragen.

SHD

Die gespeicherten Befehlsinformationen werden angezeigt.

Stop

Der Edit-Editor wird abgebrochen.

TR sw

Die Berücksichtigung nachfolgender Leerstellen wird ein- (+) oder ausgeschaltet (-).

W

Der Rest der Quelldatei wird in die Zieldatei übertragen.

Z z1

Die Endekennung für die Einfügebefehle auf z1 wird geändert.

10.3 Die CLI-/Shell-Befehle

Es werden in diesem Unterkapitel noch einmal alle CLI-/Shell-Befehle behandelt, allerdings nicht so ausführlich wie im Kapitel 2. Es wird, wie in den beiden vorangegangenen Unterkapiteln auch, eine Tabellenform abgedruckt werden. Die erste Liste wird die CLI-/Shell-Befehle, die sowohl in der Workbench-Version 1.2 als auch in der Workbench-Version 1.3 enthalten sind, nach Aufgaben sortiert ausgeben. Es wird zuerst die korrekte Syntax des Befehls der Version 1.2 beschrieben, dann folgt eine kurze Erläuterung hinsichtlich der Funktion des Befehls, und anschließend werden die möglichen Optionen erklärt. Es folgen dann die eventuell vorhandenen zusätzlichen Optionen der 1.3-Version des Befehls. Am Ende dieser Liste werden die neuen 1.3-Befehle vollständig erklärt.

Format Drive Laufwerk Namen "Name" (NoIcons)

Formatiert eine Diskette und gibt ihr einen Namen.

Laufwerk	Angabe des Laufwerks, in dem die zu formatierende Diskette liegt.
Name	Die formatierte Diskette bekommt den Namen "Name".
NoIcons	Auf der neu formatierten Diskette wird weder das Trashcan-Icon kopiert, noch das Trashcan-Verzeichnis angelegt.

V1.3 **(Quick)(FFS)(NoFFS)**

Quick	Es werden nur Root- und Boot-Blocks formatiert.
FFS	Zum Formatieren wird das FastFileSystem benutzt.
NoFFS	Das FastFileSystem wird nicht benutzt.

Dir Name (OPT (A)(I)(AI)(D)(AD))

Gibt das Inhaltsverzeichnis einer Diskette aus.

Name Gibt ein Laufwerk oder Verzeichnis an.

OPT A Zeigt alle Dateien des Verzeichnisses einschließlich enthaltener Unterverzeichnisse und deren Inhalt.

OPT I Das Inhaltsverzeichnis wird interaktiv ausgegeben. Nach jeder Datei können folgende Eingaben gemacht werden:

- *?* Ausgabe der möglichen Befehle
- *B* Es wird mit der nächsten Datei fortgefahren.
- *E* Der Inhalt der Datei wird ausgegeben.
- *Del* Die Datei wird gelöscht.
- *Q* Der Dir-Befehl wird beendet.

OPT AI Die Optionen A und I werden kombiniert

OPT D Es werden nur die Verzeichnisse aufgelistet.

OPT AD Die Optionen A und D werden kombiniert.

V1.3 **(ALL)(DIRS)(FILES)(INTER)**

ALL Siehe V1.2 OPT A.

DIRS Siehe V1.2 OPT D.

FILES Es werden nur Files, aber nicht die Unterverzeichnisse aufgelistet.

INTER Siehe V1.2 OPT I.

CD Name

Wechselt das Laufwerk bzw. das Verzeichnis.

Name Gibt das Laufwerk oder das Verzeichnis an, in das gewechselt werden soll.

MakeDir Name

Ein neues Verzeichnis namens Name wird eingerichtet.

Name Das neue Verzeichnis bekommt den Namen "Name".

Delete Name (All)(Quiet)(Q)

Löscht Dateien und/oder Verzeichnisse.

Name	Gibt die Datei an, die gelöscht wird.
All	Es wird die Datei samt komplettem Inhalt gelöscht.
Quiet	Es wird keine Löschmeldung auf dem Bildschirm ausgegeben.
Q	Siehe Quiet.

Copy Name1 (To) Name2 (All)(Quiet)

Erstellt eine Kopie einer Datei oder eines Verzeichnisses.

Name1	Name1 gibt die Quelldatei an.
To	Kann wahlweise mit eingegeben werden.
Name2	Name2 gibt die Zieldatei an.
All	Es wird die Datei samt komplettem Inhalt kopiert.
Quiet	Es wird keine Kopiermeldung auf dem Bildschirm ausgegeben.

V1.3 **(Buffer n)(Buf n)(Clone)(Date)(NoPro)(Com)**

Buffer n	Es werden beim Kopieren n 512 Byte große Puffer zur Verfügung gestellt.
Buf n	Siehe Buffer.
Clone	Datum, Status-Bits und Kommentar werden mitkopiert.
Date	Es wird das Datum mitkopiert.
NoPro	Die Status-Bits werden beim Kopieren zurückgesetzt.
Com	Der Kommentar wird mitkopiert.

Rename (From) Name1 (To)(As) Name2

Benennt eine Datei um.

From	Kann wahlweise mit eingegeben werden.
Name1	Name1 gibt Datei an, die einen anderen Namen bekommen soll.
To	Siehe From.
As	Siehe From.
Name2	Name2 gibt den neuen Namen an.

List (Name)(Pat Muster)(P Muster)(Keys)(Dates)(NoDates)
(To Name)(S Text)(Since Datum)(Upto Datum)(Quick)

Listet Daten von Dateien auf.

Name Es werden nur die Daten über das Verzeichnis oder der Datei "Name" ausgegeben.

Pat Muster Es werden nur die Daten ausgegeben, die durch Muster spezifiziert wurden.

P Muster Siehe Pat Muster.

Keys Es wird die Anzahl der File-Header-Blocks der Datei oder des Verzeichnisses ausgegeben.

Dates Datumsangabe erfolgt im Format TT-MMM-JJ.

NoDates Datumsangabe wird bei der Ausgabe unterdrückt.

To Name Ausgabe wird in die Datei Name umgelenkt.

S Text Es werden nur Daten über Dateien ausgegeben, deren Name die Buchstaben Text enthält.

Since Datum Es werden nur die Dateien berücksichtigt, die ab Datum erstellt wurden.

Upto Datum Es werden nur die Dateien berücksichtigt, die bis Datum erstellt wurden.

Quick Gibt nur die Dateinamen aus.

V1.3 **(Block)(NoHead)(Files)(Dirs)(LFormat="Text")**

Block Die File-Größe wird in Blocks angegeben.

NoHead Die Header-Informationen werden unterdrückt.

Files Es wird nur auf die Files Bezug genommen.

Dirs Es wird nur auf die Direktorys Bezug genommen.

LFormat="Text"

Die Option LFormat bewirkt, daß der in Text enthaltene Text ausgegeben wird. Bei Eingabe von einem %s gilt das %s als Platzhalter für den aktuellen Dateinamen. Bei Eingabe von zwei %s wird der aktuelle Dateiname zweimal ausgegeben. Bei Eingabe von drei %s steht das erste %s stellvertretend für die Pfadangabe der aktuellen Datei. Das zweite und dritte %s enthalten den Dateinamen. Bei Eingabe von vier %s steht das erste und dritte %s für die Pfadangabe und das zweite und vierte %s für den Dateinamen.

DiskCopy (From) Laufwerk1 To Laufwerk2 (NAME Name)

Erstellt eine Kopie einer Diskette.

From Kann wahlweise mit eingegeben werden.
Laufwerk1 Laufwerk1 gibt das Quellaufwerk an.
Laufwerk2 Laufwerk2 gibt das Ziellaufwerk an.
NAME Name Name gibt den Namen an, den die Kopie erhalten soll.

ReLabel Laufwerk Name

Benennt den Namen einer Diskette um.

Laufwerk Gibt das Laufwerk an, in dem die Diskette liegt, deren Name geändert werden soll.
Name Gibt den Namen an, den die Diskette erhalten soll.

Info

Gibt Informationen über die angeschlossenen Laufwerke auf dem Bildschirm aus.

V1.3 **(Device)**
Device Es wird ein Device spezifiziert.

Install Laufwerk

Aus einer Leerdiskette wird eine bootfähige Diskette gemacht.

Laufwerk Gibt an, in welchem Laufwerk zu installierende Diskette liegt.

V1.3 **(NoBoot)(Check)**
NoBoot Die Diskette wird nicht bootfähig gemacht.
Check Es wird geprüft, ob Diskette bootfähig ist und ob ein Standard-Boot-Code vorhanden ist.

Type (From) Name1 ((To) Name2)(OPT(N)(H))

Gibt den Inhalt einer Datei aus.

From Kann wahlweise mit eingegeben werden.
Name1 Name1 gibt die Quelldatei an.

To Siehe From.

Name2 Name2 gibt die Zieldatei an, in die Name1 kopiert wird. Ist keine Zieldatei angegeben, wird die Ausgabe auf den aktuellen Ausgabekanal geleitet.

OPT N Die ausgegebenen Zeilen werden mit Zeilennummern versehen.

OPT H Die Zeichen werden als Hex-Zahl ausgegeben.

V1.3 **(To)(Hex)(Number)**

To Wird To in Verbindung mit Name2 eingegeben, so wird die Datei Name2, wenn sie schon existiert, ohne Abfrage überschrieben. Wird To hingegen weggelassen, wird der Type-Vorgang mit einer Meldung abgebrochen, sofern die Datei Name2 schon existiert.

Hex Siehe V1.2 OPT H.

Number Siehe V1.2 OPT N.

Join Name1 Name2 AS Name3

Faßt bis zu 15 Dateien zu einer Datei zusammen.

Name1 Name1 gibt eine Datei an, die mit anderen Dateien zusammengefaßt werden soll.

Name2 Name2 gibt eine weitere Datei an.

Name3 Name3 gibt die Datei an, in der die zusammengefaßten Dateien geschrieben werden sollen.

V1.3 **(To)**

TO Es besteht die Möglichkeit, anstelle von AS auch TO einzugeben.

Search (From) Name (Search) Text (All)

Sucht in Dateien nach einer angegebenen Zeichenkette.

From Kann wahlweise mit eingegeben werden.

Name Gibt die Datei an, in der gesucht werden soll.

Search Siehe From.

Text Gibt Zeichenkette an, die gesucht werden soll.

All Es wird auch in den Unterverzeichnissen von Name gesucht.

V1.3 **(NoNum)(Quiet)(Quick)(File)**

NoNum Falls die gesuchte Zeichenkette gefunden wurde, wird keine Zeilennummer ausgegeben.

Quiet Es erfolgt keine Ausgabe.

Quick Das Ausgabeformat wird kompakter.

File Es wird zuerst das spezifizierte File gesucht und dann erst nach der Zeichenkette.

Sort (From) Name1 (To) Name2 (Colstart n)

Sortiert Datei alphabetisch und speichert sie in einer anderen ab.

From Kann wahlweise mit eingegeben werden.

Name1 Gibt die Quelldatei an.

To Siehe From.

Name2 Gibt an, in welche Datei sortierter Text geschrieben werden soll.

Colstart n Gibt an, ab welcher Zeile der Text sortiert werden soll.

Protect (File) Name (Flags) Status

Bestimmt, in welcher Weise auf eine Datei zugegriffen werden kann.

File Kann wahlweise mit eingegeben werden.

Name Gibt an, für welche Datei der Status gesetzt wird.

Flags Siehe File.

Status Gibt die Protektion an. Folgende Eingaben sind möglich:

- *R* Auf die Datei darf lesend zugegriffen werden.
- *W* Auf die Datei darf schreibend zugegriffen werden.
- *E* Die Datei ist ausführbar.
- *D* Die Datei ist löschbar.

V1.3 **(+)(-)(Add)(Sub)**

+ Es wird das bei Status angegebene Status-Bit gesetzt.

- Es wird das bei Status angegebene Status-Bit zurückgesetzt.

Add Siehe +.

Sub Siehe -.
Ab V1.3 können noch das Hidden- (H), Script- (S), Pure- (P) und Archive- (A) Bit gesetzt oder zurückgesetzt werden.
H Die Datei wird versteckt (unsichtbar).
S Die Datei kann ohne execute gestartet werden (gilt nur für Batch-Dateien).
P Die Datei kann in die Residentliste aufgenommen werden.
A Die Datei wird bei einem Backup (Copy-Befehl) nicht berücksichtigt, wenn es gesetzt ist. Das H- und A-Bit funktioniert nur mit Kickstart 1.3.

FileNote (File) Name (Comment) Text

Versieht eine Datei mit einem Kommentar.

File Kann wahlweise mit eingegeben werden.
Name Gibt an, welche Datei den Kommentar erhält.
Comment Siehe File.
Text Ist der Kommentar, den die Datei erhält.

SetDate (File) Name (Date) Datum ((Time) Zeit)

Versieht eine Datei mit einer Datums- bzw. Zeitangabe.

File Kann wahlweise mit eingegeben werden.
Name Gibt an, welche Datei das Datum bzw. die Zeit erhalten soll.
Date Siehe File.
Datum Enthält das Datum, das die Datei erhalten soll.
Time Kann wahlweise in Verbindung mit Zeit eingegeben werden.
Zeit Enthält die Zeitangabe, die die Datei erhalten soll.

DiskDoctor (Drive) Laufwerk

Eventuell werden Fehler an Disketten oder Dateien behoben.

Drive Kann wahlweise mit eingegeben werden.

Laufwerk Gibt an, auf welches Laufwerk der DiskDoktor zugreifen soll.

DiskChange Laufwerk

Übergibt dem AmigaDOS die Mitteilung, daß eine Diskette gewechselt wurde.

Laufwerk Gibt an, in welchem Laufwerk die Diskette gewechselt wurde.

NewCLI (Con:x/y/Breite/Höhe(/Text))(From Name)

Eröffnet ein neues CLI.

x Gibt x-Position der linken oberen Ecke des neuen Windows an.
y Gibt y-Position der linken oberen Ecke des neuen Windows an.
Breite Gibt die Breite des Windows in Pixeln an.
Höhe Gibt die Höhe des Windows in Pixeln an.
Text Gibt den Titel des neuen Windows an.
From Name Nachdem das neue CLI geöffnet wurde, wird automatisch die Batch-Datei Name abgearbeitet.

EndCLI

Schließt ein CLI.

Run Befehl

Arbeitet einen Befehl im Hintergrund ab.

Befehl Enthält einen AmigaDOS-Befehl, der im Hintergrund abgearbeitet wird.

Status (Process) Nummer (Full)(TCB)(CLI)(All)

Gibt Informationen über CLI-Prozesse aus.

Process Kann wahlweise mit eingegeben werden.
Nummer Gibt an, über welchen Prozeß die Informationen ausgegeben werden sollen.

Full Kombiniert die Optionen TCB und CLI.
TCB Es werden Informationen über Priorität und Stack-Größe des Prozesses ausgegeben.
CLI Gibt die momentan aufgerufenen Befehle der Prozesse aus.
All Siehe CLI.

V1.3 **(Com=Befehl)(Command=Befehl)**
Com=Befehl Es wird nach dem CLI-Befehl Befehl gesucht.
Command=BefehlSiehe Com=Befehl.

ChangeTaskPri (Pri) n

Ordnet einem Prozeß eine Priorität zu.

Pri Kann wahlweise mit eingegeben werden.
n Enthält die neue Priorität.

V1.3 **(Process n)**
Process n Die neue Priorität wird dem Prozeß Nummer n zugewiesen.

Break (Task) Nummer (All)(C)(D)(E)(F)

Unterbricht einen Prozeß.

Task Kann wahlweise mit eingegeben werden.
Nummer Verweist auf den zu unterbrechenden Prozeß.
All Es werden die Unterbrechungsebenen C, D, E und F gesetzt.
C Es wird die Unterbrechungsebene C gesetzt.
D Es wird die Unterbrechungsebene D gesetzt.
E Es wird die Unterbrechungsebene E gesetzt.
F Es wird die Unterbrechungsebene F gesetzt.

V1.3 **(Process)**
Process Siehe V1.2 Task.

Path (Name ADD)(Show)((Name) Reset)

Zeigt oder ändert die Verzeichnispfade.

Name ADD Es wird zusätzlich ein Pfad zum Verzeichnis "Name" gelegt.

Show Show zeigt die aktuellen Pfade an.
Name Reset Alle Pfade bis auf C- und Name-Pfad werden gelöscht.

V1.3 **(Quiet)**
Quiet Es erfolgen keine Ausgaben auf dem aktuellen Ausgabekanal.

Assign ((Name) Name1 ((Dir) Name2))(List)

Weist einem logischen Gerät ein Verzeichnis zu.

Name Kann wahlweise mit eingegeben werden.
Name1 Gibt das zuzuweisende logische Gerät an.
Dir Siehe Name.
Name2 Gibt das Verzeichnis an, dem das logische Gerät zugewiesen wird.
List Es werden die aktuellen logischen Zuweisungen aufgelistet.

V1.3 **(Exists Name)(Dismount Name)(Unmount Name)**
Exists Name Es wird in der Assign-Liste nach Name gesucht. Ist er nicht vorhanden, wird ein Fehlercode von 5 zurückgegeben, ansonsten wird kein Fehlercode zurückgegeben.
Dismount Name Name wird aus der Assign-Liste entfernt.
Unmount Name Siehe Dismount Name.

AddBuffers (Drive) Laufwerk (Buffers) n

Einem Laufwerk wird ein bestimmter Speicherplatz reserviert.

Drive Kann wahlweise mit eingegeben werden.
Laufwerk Gibt Laufwerk an, das den Puffer erhalten soll.
Buffers Siehe Drive.
n Gibt die Größe des zu reservierenden Puffers in Blocks an.

Why

Gibt Informationen über den zuletzt aufgetretenen Fehler aus.

Fault n

Gibt die Fehlermeldung zum entsprechenden Fehlercode aus.

n Gibt einen gültigen Fehlercode an.

Date (Time) Zeit (Date) Datum (To Name)(Ver Name)

Eingabe oder Ausgabe von Datum und/oder Zeit.

Time Kann wahlweise mit eingegeben werden.
Zeit Enthält die einzugebende Uhrzeit.
Date Siehe Time.
Datum Enthält das einzugebende Datum.
To Name Name gibt eine Datei an, in die das Datum sowie die Uhrzeit geschrieben werden soll.
Ver Name Siehe To Name.

SetClock (OPT Load)(OPT Save)

Übergabe von Datum und Zeit des Systems an die Uhr und umgekehrt.

OPT Load Datum und Zeit werden von der internen Uhr als Systemzeit übernommen.
OPT Save Das Systemdatum und die Systemzeit werden an die interne Uhr übergeben.

V1.3 **(Load)(Save)**

Load Datum und Zeit werden von der internen Uhr als Systemzeit übernommen.
Save Das Systemdatum und die Systemzeit werden an die interne Uhr übergeben.

Prompt Text

Ändert die Bereitschaftsanzeige der CLI-Prozesse.

Text Enthält das Aussehen der Bereitschaftsanzeige, wobei %n die CLI-Prozeßnummer angibt.
V1.3 In Verbindung mit der neuen Shell kann auch %s verwendet werden. Es wird dann auch das aktuelle Verzeichnis ausgegeben.

Stack (n)

Verändert die Stack-Größe oder gibt die aktuelle Größe aus.

n Gibt die Stack-Größe in Bytes an.

BindDrivers

Es werden weitere Treiber ins System eingebunden.

Mount (Device) Name

Bereitstellen eines Devices.

Device Kann wahlweise mit eingegeben werden.
Name Gibt den Namen des neuen Devices an.

V1.3 **(From Name)**

From Name Die Parameter werden nicht aus der Datei Devs/Mountlist genommen, sondern aus der Datei Name.

Execute Name (Text)

Arbeitet eine Batch-Datei ab.

Name Gibt den Namen der Batch-Datei an.
Text Enthält Argumente, die an die Batch-Datei übergeben werden.

Echo "Text"

Gibt einen Text auf den aktuellen Ausgabekanal aus.

"Text" Enthält einen Text, der auf dem Ausgabekanal ausgegeben werden soll.

V1.3 **(NoLines)**

NoLines Nach Ausgabe des angegebenen Strings wird nicht in eine neue Zeile gesprungen.

Failat (n)

Setzt die Abbruchgrenze fest oder gibt die aktuelle Abbruchgrenze aus.

n Enthält die Größe der neuen Abbruchgrenze.

Quit (n)

Abbruch einer Batch-Datei mit eventueller Übergabe eines Fehlercodes.

n Enthält einen Fehlercode.

If (Not)(Warn)(Error)(Fail)(Text1 EQ Text2)(Exists Name) Befehl1 (Else Befehl2) EndIf

Der Befehl entscheidet, ob bestimmte Befehle innerhalb einer Batch-Datei abgearbeitet werden oder nicht.

Not Bewirkt die logische Umkehrung einer Bedingung.

Warn Bedingung erfüllt, wenn Fehlercode größer oder gleich 5.

Error Bedingung erfüllt, wenn Fehlercode größer oder gleich 10.

Fail Bedingung erfüllt, wenn Fehlercode größer oder gleich 20.

Text1 EQ Text2 Bedingung erfüllt, wenn Text1 gleich Text2 ist.

Exists Name Bedingung erfüllt, wenn die Datei Name vorhanden ist.

Befehl1 CLI-Befehl, der ausgeführt wird, wenn Bedingung erfüllt wurde.

Else Befehl2 CLI-Befehl, der ausgeführt wird, wenn die Bedingung nicht erfüllt wurde.

Skip Text

Es wird innerhalb einer Batch-Datei zu einer Marke gesprungen.

Text Enthält eine Zeichenkette, zu der verzweigt wird, wenn sie als Label definiert ist.

Lab Text

Lab definiert eine Zeichenkette als Sprungmarke.

Text	Enthält Zeichenkette, die als Label definiert wird.

Ask "Text"

Abfrage auf (Y)es oder (N)o, wobei y ein Fehlercode von 5, und n keinen Fehlercode ergibt.

"Text"	Enthält Text, der auf dem aktuellen Ausgabekanal ausgegeben wird.

Wait (n)(Sec)(Secs)(Min)(Mins)(Until Zeit)

Wait versetzt das System in den Wartezustand.

n	Gibt die Anzahl der Einheiten an, die gewartet werden soll.
Sec	Spezifiziert die Einheit als Sekunde.
Secs	Siehe Sec.
Min	Spezifiziert die Einheit als Minute.
Mins	Siehe Min.
Until Zeit	Es wird gewartet, bis die in Zeit enthaltene Uhrzeit erreicht ist.

Die neuen CLI-Befehle der Workbench 1.3

Alias Name String

Dieser Befehl kann nur in Verbindung mit der Shell benutzt werden. Dabei wird einem Wort ein String zugewiesen. (Siehe Kapitel 6)

Name	Das neue Befehlswort.
String	Enthält die Befehlszeile, die mit Name aufgerufen wird.

Avail

Es wird ein Überblick über den vorhandenen Speicherplatz ausgegeben.

FF (-0)(-n)

Die Textausgabe auf dem Bildschirm wird beschleunigt.

-0 FastFont-Textausgabe wird eingeschaltet.
-n FastFont-Textausgabe wird ausgeschaltet.

Getenv Name

Es wird der Inhalt einer Variablen gelesen.

Name Gibt die Bezeichnung der Variablen an, deren Inhalt gelesen werden soll.

Lock Laufwerk (On Passwort)(Off Passwort)

Blockiert oder gibt den Zugriff auf eine Festplatten-Partition frei.

Laufwerk Enthält die zu schützende Festplatten-Partition.
On Passwort Der Zugriff auf die Festplatten-Partition wird blockiert. Nach Eingabe des Paßwortes (max. 4 Zeichen) wird der Zugriff für diese Benutzung wieder freigegeben.
Off Passwort Ein Paßwort kann mit dieser Eingabe wieder aufgehoben werden. In Passwort muß das entsprechende Paßwort vorhanden sein.
Dieser Befehl funktioniert nur mit Kickstart 1.3

NewShell (Newcon:x/y/Breite/Höhe(/Text))(From Name)

Eröffnet eine neue Shell.

x Gibt x-Position der linken oberen Ecke des neuen Windows an.
y Gibt y-Position der linken oberen Ecke des neuen Windows an.
Breite Gibt die Breite des Windows in Pixeln an.
Höhe Gibt die Höhe des Windows in Pixeln an.
Text Gibt den Titel des neuen Windows an.
From Name Nachdem die neue Shell geöffnet wurde, wird nicht das Shell-Startup-File aus dem S-Verzeichnis, sondern das File Name automatisch abgearbeitet.

Remrad

Es werden alle Dateien aus der resetfesten RAM-Disk gelöscht. Außerdem wird nach dem nächsten Booten das Ramdrive.Device aus dem System entfernt.

Resident Name Datei (Remove)(Add)(Replace)(Pure)(System)

Es wird ein neuer Befehl in die Liste der residenten Befehle aufgenommen, gelöscht oder ersetzt.

Name	Kann wahlweise mit eingegeben werden.
Datei	Enthält den Befehl, der in die Resident-Liste aufgenommen werden soll.
Remove	Der Befehl wird aus der Liste gelöscht.
Add	Der Befehl wird in die List aufgenommen.
Replace	Ist ein Befehl mit dem angegebenen Namen in der Liste schon vorhanden, so wird er durch den neuen ersetzt.
Pure	Es wird überprüft, ob das Pure-Bit des Befehls gesetzt ist oder nicht.
System	Benutzen Sie diese Option nicht, es sei denn, es wird speziell angegeben.

SetAlert

Das Betriebssystem wird gepatched.

Setenv Name String

Es wird einer Variablen ein String zugewiesen.

Name	Die Bezeichnung der Variablen.
String	Enthält eine Zeichenkette, die der Variablen zugewiesen wird.

Which Befehl

Es wird der Pfad zu einem Befehl gesucht und ausgegeben.

Befehl	Enthält den zu suchenden Befehl.

XIcon

Eine Batch-Datei kann mit Hilfe dieses Befehls auch von der Workbench aus mit der Maus aktiviert werden (Siehe Kapitel 6).

10.4 Alle CLI-Befehle alphabetisch geordnet

Diese Liste beinhaltet alle CLI-/Shell-Befehle alphabetisch geordnet. Auch hier ergänzt eine kurze Erläuterung den einzelnen Befehl.

AddBuffers ..... *Seite 105*

Einem Laufwerk wird ein bestimmter Speicherplatz reserviert.

Alias ..... *Seite 241*

Dieser Befehl kann nur in Verbindung mit der Shell benutzt werden. Dabei wird einem Wort ein String zugewiesen (Siehe Kapitel 6).

Ask ..... *Seite 127*

Abfrage auf (y)es oder (n)o, wobei y einen Fehlercode von 5 und n keinen Fehlercode ergibt.

Assign ..... *Seite 102*

Weist einem logischen Gerät ein Verzeichnis zu.

Avail ..... *Seite 158*

Es wird ein Überblick über den vorhandenen Speicherplatz ausgegeben.

BindDrivers ..... *Seite 112*

Es werden dem System weitere Treiber hinzugefügt.

Break ..... *Seite 99*

Unterbricht einen Prozeß.

CD ..... *Seite 38*
Wechselt das Laufwerk bzw. das Verzeichnis.

ChangeTaskPri ..... *Seite 97*
Ordnet einem Prozeß eine Priorität zu.

Copy ..... *Seite 44*
Erstellt eine Kopie einer Datei oder eines Verzeichnisses.

Date ..... *Seite 108*
Ein- oder Ausgabe von Datum und/oder Zeit.

Delete ..... *Seite 42*
Löscht Dateien und/oder Verzeichnisse.

Dir ..... *Seite 34*
Gibt das Inhaltsverzeichnis einer Diskette aus.

DiskChange ..... *Seite 86*
Übergibt dem AmigaDOS die Mitteilung, daß eine Diskette gewechselt wurde.

DiskCopy ..... *Seite 58*
Erstellt eine Kopie einer Diskette.

DiskDoctor ..... *Seite 80*
Es werden Fehler an Disketten oder auf Dateien evtl. behoben.

Echo ..... *Seite 120*
Gibt einen Text auf dem aktuellen Ausgabekanal aus.

EndCLI ..... *Seite 93*
Schließt ein CLI.

Execute ..... *Seite 117*
Arbeitet eine Batch-Datei ab.

Failat ..... *Seite 121*
Setzt die Abbruchgrenze fest oder gibt die aktuelle Abbruchgrenze aus.

Fault ..... *Seite 107*
Gibt die Fehlermeldung zum entsprechenden Fehlercode aus.

FF ..... *Seite 159*
Die Textausgabe auf dem Bildschirm wird beschleunigt.

FileNote ..... *Seite 78*
Versieht eine Datei mit einem Kommentar.

Getenv ..... *Seite 169*
Es wird der Inhalt einer Variablen gelesen.

If (Else) EndIf ..... *Seite 123*
Entscheidet, ob bestimmte Befehle innerhalb einer Batch-Datei abgearbeitet werden oder nicht.

Info ..... *Seite 61*
Gibt Informationen über die angeschlossenen Laufwerke aus.

Install ..... *Seite 63*
Macht aus einer Leerdiskette eine bootfähige Diskette.

Join ..... *Seite 70*
Faßt zwei oder mehrere Dateien zu einer Datei zusammen.

Lab ..... *Seite 128*
Definiert eine Sprungmarke.

List ..... *Seite 48*
Listet Informationen über Dateien auf.

Lock ..... *Seite 160*
Blockiert oder gibt den Zugriff auf ein Laufwerk frei.

MakeDir ..... *Seite 40*
Richtet ein neues Verzeichnis ein.

Mount ..... *Seite 113*
Stellt ein Device bereit.

NewCLI ..... *Seite 87*
Eröffnet ein neues CLI.

NewShell ..... *Seite 160*
Eröffnet eine neue Shell.

Path ..... *Seite 100*
Zeigt oder ändert die Verzeichnispfade.

Prompt ..... *Seite 110*
Ändert die Bereitschaftsanzeige der CLI-Prozesse.

Protect ..... *Seite 74*
Bestimmt die Zugriffsmöglichkeiten auf eine Datei.

Quit ..... *Seite 123*
Abbruch einer Batch-Datei mit eventueller Übergabe eines Fehlercodes.

ReLabel ..... *Seite 60*
Benennt den Namen einer Diskette um.

Remrad ..... *Seite 164*
Es wird die resetfeste RAM-Disk gelöscht.

Rename ..... *Seite 56*
Benennt eine Datei um.

Resident ..... *Seite 164*
Es wird ein neuer Befehl in die Liste der residenten Befehle aufgenommen, gelöscht oder ersetzt.

Run ..... *Seite 93*
Arbeitet einen Befehl im Hintergrund ab.

Search ..... *Seite 71*
Sucht in Dateien nach einer angegebenen Zeichenkette.

SetAlert ..... *Seite 168*
Das Betriebssystem wird gepatched.

SetClock ..... *Seite 109*
Übergabe von Datum und Zeit des Systems an die Uhr und umgekehrt.

SetDate ..... *Seite 79*
Versieht eine Datei mit einer Datums- bzw. einer Zeitangabe.

Setenv ..... *Seite 169*
Es wird einer Variablen ein String zugewiesen.

Skip ..... *Seite 128*
Springt innerhalb einer Batch-Datei zu einer Marke.

Sort ..... *Seite 73*
Sortiert eine Datei alphabetisch und speichert sie in einer anderen Datei ab.

Stack ..... *Seite 111*
Verändert die Stack-Größe oder gibt die aktuelle Größe aus.

Status ..... *Seite 95*
Gibt Informationen über CLI-Prozesse aus.

Type ..... *Seite 68*

Gibt den Inhalt einer Datei aus.

Wait ..... *Seite 129*

Versetzt das System in den Wartezustand.

Which ..... *Seite 353*

Es wird der Pfad zu einem Befehl gesucht und ausgegeben.

Why ..... *Seite 106*

Gibt Informationen über den zuletzt aufgetretenen Fehler aus.

XIcon ..... *Seite 169*

Eine Batch-Datei kann mit Hilfe dieses Befehls auch von der Workbench aus mit der Maus aktiviert werden (Siehe Kapitel 6).

Anhang A

Steuer- und Editorsequenzen im CLI/Shell

Unter Verwendung der Ctrl- und Esc-Tasten können bestimmte Sequenzen ins CLI/Shell direkt oder durch den Echo-Befehl innerhalb einer Batch-Datei eingegeben werden. Wenn der Echo-Befehl benutzt wird, dann kann die Esc-Taste durch die Zeichenkombination *e ersetzt werden.

Escapesequenzen

<Esc>c	Der Inhalt des CLI-/Shell-Fensters wird gelöscht, und es werden alle Sondermodi abgeschaltet.
<Esc>[0m	Es werden alle Sondermodi abgeschaltet.
<Esc>[1m	Fettschrift wird eingeschaltet.
<Esc>[2m	Farbnummer2 wird zur Schriftfarbe (Schwarz).
<Esc>[3m	Kursivschrift wird eingeschaltet.
<Esc>[30m	Farbnummer0 wird zur Schriftfarbe (Blau).
<Esc>[31m	Farbnummer1 wird zur Schriftfarbe (Weiß).
<Esc>[32m	Farbnummer2 wird zur Schriftfarbe (Schwarz).
<Esc>[33m	Farbnummer3 wird zur Schriftfarbe (Orange).
<Esc>[4m	Der Text wird unterstrichen.
<Esc>[40m	Farbnummer 0 wird zur Hintergrundfarbe (Blau).
<Esc>[41m	Farbnummer 1 wird zur Hintergrundfarbe (Weiß).
<Esc>[42m	Farbnummer 2 wird zur Hintergrundfarbe (Schwarz).
<Esc>[43m	Farbnummer 3 wird zur Hintergrundfarbe (Orange).
<Esc>[7m	Der Text wird invers dargestellt.
<Esc>[8m	Der Text wird unsichtbar (Blau).
<Esc>[nu	Das CLI-/Shell-Fenster wird n Zeichen breit.
<Esc>[nt	Die Anzahl der Zeilen im CLI-/Shell-Fenster wird auf n gesetzt.
<Esc>[nx	Der linke Rand wird auf n Pixel gesetzt.
<Esc>[ny	Der Abstand von oben wird auf n Pixel gesetzt.

Control-Sequenzen

Bei der Eingabe von Control-Sequenzen muß die Ctrl-Taste und die entsprechende Buchstabentaste gleichzeitig gedrückt werden.

<Ctrl>-<h> Das zuletzt eingegebene Zeichen wird gelöscht.
<Ctrl>-<i> Der Cursor wird um eine Tabulatorposition nach rechts bewegt.
<Ctrl>-<j> Bewirkt einen Zeilenvorschub.
<Ctrl>-<k> Der Cursor wird um eine Zeile nach oben bewegt.
<Ctrl>-<l> Der Inhalt des CLI-/Shell-Fensters wird gelöscht.
<Ctrl>-<m> Bewirkt das gleiche wie Return.
<Ctrl>-<o> Es wird der normale Zeichensatz eingeschaltet.
<Ctrl>-<n> Es wird der ALT-Zeichensatz eingeschaltet.
<Ctrl>-<x> Die gesamte aktuelle Zeile wird gelöscht.
<Ctrl>-<\> Kennzeichnet das Ende einer Datei.

Index

.bra 119
.def 118
.dol 119
.dot 120
.info 22
.ket 120
.key 117

60 Zeichen pro Zeile 64

AddBuffers 347, 354
Aktuelles Inhaltsverzeichnis 35
Aktueller Ordner 38
Alias 163, 241, 354
AmigaDOS 16
Anhalten der Ausgabe 260
Anzeige des Befehls-Status 268
Argument 26
ASCII 197
Ask 217, 351, 354
Assign 347, 354
Ausgabe einer Zeichenkette 120
Aux-Device 174
Avail 158, 354

Backspace 20
Backup-Befehl 225
Batch-Befehle 225
Batch-Dateien 88, 206, 211
Batch-Files 65, 205
Batch-Variablen 124
Batchprocessing 205
Baud 151
Baumstruktur 22
Beenden 28
Befehlseinheiten 25

Belegungsgrad 62
Bildschirm 154
BindDrivers 349, 354
Bootbar 63
Bootbare Diskette 65
BPTR-Pointer 279
Break 260, 346, 354
BSTR 288

CD 38, 338, 355
CD / 39
ChangeTaskPri 265, 346, 355
CHIP-Memory 158
CLI 17
CLI-/Shell-Befehle 206, 337
CLI-/Shell-Fenster 217
CLI-Befehle 157
CLI-Fenster 206
CLI-Kommandos 325
CLI-Startup-Dateien 221
CLI-Wecker 196
Command Line Interface 17
Con-Device 154, 193
Copy 339, 355
Ctrl und C 59, 260

Date 348, 355
Debug 201
Defekte Blocks 62
Delete 339, 355
Dir 21, 22, 338, 355
Direkte Befehle 326
DiskChange 345, 355
DiskCopy 341, 355
DiskDoctor 344, 355
Diskette retten 252
DOS 15
DOS-Prompt 20
Dringlichkeitsstufe 97
Drucken vom CLI aus 188

Drucker 189
Drucker-Batch-Datei 222
Drucker-Device 153
Druckereinstellungen 223
Duplikat einer Diskette 58

Echo 349, 355
Eckige Klammern 193
ED-Editor 132, 325, 208
ED-Funktionen 325
Edit 132, 325
Edit-Funktionen 328
Eigene Fenster 194
Ein- und Ausgaben 177
Eingabeparameter 26
Else 350
EndCLI 345, 351, 355
EndIf 350, 356
Enter-Taste 20
EQ 124
ERROR 125
Escape-Sequenzen 222, 243
Execute 349, 355

Failat 350, 356
Fast-File-System 174
FAST-Memory 158
FastFilingSystem 160
FastFonts 159
Fault 348, 356
Fehlergrenze 122
Fehlernummer 121
Fenster 154
FF 159, 175, 356
File-Ausdruck 188
FileNote 344, 356
Format 337
Format-Zeile 25
Formatieren 30
Fragezeichen 179

Gerätetreiber 247
Getenv 169, 356
Global-Vektor 286
Global-Vector-Tabelle 96, 288
Größe des voreingestellten CLI-Fensters 253
Guru-Meditation 201

Handshake-Problem 151
Hauptverzeichnis 22, 38
Hilfsfunktion 25
Hintergrundprozeß 93
Hochfahren 211

If 350, 356
Info 61, 341, 356
Initialize 30
Install 341, 356
Interne DOS-Bibliothek 284

Join 342, 356
Jokerzeichen 178

Key 26
Keymaps 65
Komfortable Startup-Sequence 216
Kommandomodus 325
Kommandomodus-Befehle 327
Kopie 18
Kopieren 44
Kopieren mit der RAM-Disk 187
Kopieren von mehreren Dateien 252
Kopiervorgang 44

Lab 128, 351, 356
Laufwerksbezeichnung 24
Leerzeichen 31
List 340, 356
Loadwb -debug 201
Lock 160, 357
Löschen von mehreren Dateien 253

MakeDir 338, 357
Modem-Betrieb 199
Mount 61, 349, 357
Multitasking 247
Multiuser-Betrieb 197

Namenskollisionen 45
Neue CLI 264
Neustart 211
NewCLI 254, 345, 357
NewCon-Device 170
NewShell 160, 357
NOICONS 31
NOT 126

Ordner 38
Ordner umbenennen 57
Ordnernamen 23

Papierkorb 31
Paralleles Device 150
Parameter 26
Path 346, 357
Pfadangabe 23, 24, 38
Pfade 41
Pfade hinzuzufügen 101
Pipe-Device 172
Priorität 96, 97, 265, 272
Problemloses Arbeiten 268
Programm CLI 282
Prompt 348, 357
Protect 343, 357
Prozessor-Stacks 96
Prt-Device 153
Pufferspeicher 105
Pure-Flag 166

Quelldatei 44
Quellordner 44
Quit 350, 357

RAD-Device 171
RAM 251
RAM-Disk 148, 183, 227
Raw-Device 155
ReLabel 341, 357
Relokatible Programme 279
Remrad 164, 357
Rename 339, 357
Resetfeste Ram-Disk 164
Resident 164, 358
Return-Taste 20
ROM-Wack 201
RS 232C-Schnittstelle 151
Run-Befehl 257, 288, 345, 358

Schnellere Startup-Sequence 216
Schnittstelle 150
Schrägstrich 24, 35
Schreibmaschine 194
Schreibschutz 18
Schublade 22, 40
Search 342, 358
Segmente 279, 288
Segmentliste 281
Sektoren 32
Serielles Device 151
Serielle Schnittstelle 197
SetAlert 168, 358
SetClock 348, 358
SetDate 344, 358
Setenv 169, 358
Setmap d 65
Shell 160
Shell-Startup-Dateien 221
Skip 128, 350, 358
Sort 343, 358
Speak-Device 173
Speicherkapazität 32
Spezielle Tastaturbelegungen 65
Stack 349, 358

Stapeldateien 116
Stapeldateiverarbeitung 205
Stapelspeicher 111
Startup-Sequence 66, 209, 219
Status 345, 358
Status Full 269
Status-Befehl 261
Statusinformation 50
Steuerzeichen 191, 222
Switch 26
Symbol des CLI 19
System-Configuration 64

Task Held 201, 252
Tastatur 154
Textdateien auf dem Bildschirm ausgeben 68
Textdateien erstellen 195
Textdateien erweitern 195
Titel des neuen Fensters 88
Type 341, 359

Umlaute 65
Umleiten 121, 248
Umlenkung der Ausgaben 69
Umrechnung 197
Unterbrechungen 180
Unterverzeichnis 22. 35, 38, 66

Variablen 118
Verkettung 281
Verlassen eines Batch-Files 123
Verzeichnisstruktur 22
Vorbereitung 17

Wait 129, 351, 359
Warmstart 218
WARN 125
Wartezeiten 272
Which 359
Why 347, 359

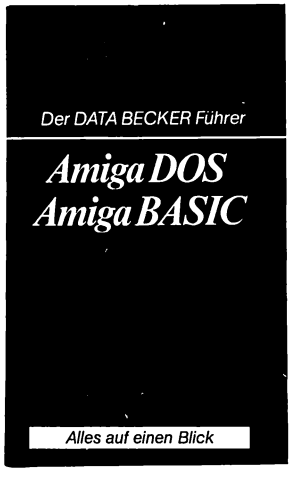
Window 226
Workbench 1.3 158
Workbench-Diskette 209

XIcon 169, 236, 359

Zeitangaben 80
Zeitersparnis 212
Zielpfad 44, 46
Zylinder 32

Hilfe

In der Regel klappt sie phantastisch, die Arbeit mit dem Computer. Und für Zweifelsfälle hat man ja bereits eine ansehnliche Bibliothek nützlicher Literatur. Doch immer wieder – mitten in der Arbeit – passiert es: Man sucht nach einem bestimmten Kommando. Irgendwo im Handbuch, oder stand es in einem Computermagazin... Der Arbeitsfluß ist unterbrochen. Man versucht sich zu erinnern, durchwühlt den riesigen Literaturberg, sucht einen Hinweis. HILFE. Genau die bekommen Sie von den neuen DATA BECKER Führern. Ein gezielter Griff und Sie haben die gewünschte Information. Hier finden sie umfassend alles auf einem Blick. Zu Ihrem Rechner oder auch zur entsprechenden Software. Das sind die ersten DATA BECKER Führer:

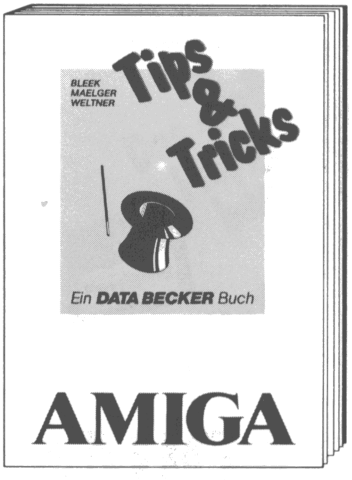


Der DATA BECKER Führer zu Amiga DOS Amiga BASIC

272 Seiten
DM 24,80

Bücher zum AMIGA

Amiga Tips und Tricks ist eine riesige Fundgrube für den Amiga-Besitzer. Viele Beispielprogramme in BASIC und C zeigen, wie man die fantastischen Möglichkeiten dieses Superrechners optimal nutzen kann. Und ganz nebenbei lernt man noch eine Menge über den Aufbau des Computers und seine Programmierung.



Aus dem Inhalt:

- Nutzung der wichtigsten Libraries von BASIC aus: Graphic, DOS, Exec, Intuition
- Nutzung der verschiedenen Disk-Fonts in BASIC-Programmen
- Verschiedene Schrifttypen in BASIC-Programmen: Bold, Outline, Shadow
- Zugriff auf das CLI von BASIC aus
- Bewegbare Screens und Windows mit eigenen Titeln
- Intuition in eigenen Programmen nutzen: Autorequest, Guru Meditation
- Gesamte Directory-Struktur ausdrucken
- Ein-/Ausgabehandling: Diskmonitor, Hardcopy von Windows und Screen
- Speicherverwaltung: AllocMem und FreeMem
- Filehandling in C: Anzahl freier Blöcke, File exist Prüfung, Filegröße, File-kommentar, Get Protection Prüfung
- Zugriff auf Intuition am Beispiel eines einfachen Grafikprogramms: Screen, Windows, Menue
- Half Bright und Interlace Modus
- Druckerhandling in C

Weltner/Hornig
AMIGA Tips & Tricks
Hardcover, 473 Seiten, DM 49,–
ISBN 3-89011-211-0

Bücher zum Amiga

Dieses DATA BECKER Floppybuch beschreibt alles Wissenswerte zum Thema Floppy ausführlich und sofort nutzbar. Zusätzlich finden Sie in diesem Buch Insider-Informationen über Kopierschutz, Speed-Routinen und Viren, die in einschlägigen Kreisen für Aufsehen sorgen werden. Daß diese Informationen in einfach zu bedienende Programme umgesetzt wurden, macht dieses Buch zu einer unersetzlichen Hilfe bei der täglichen Arbeit mit dem Amiga.

Aus dem Inhalt:

- Floppy-Operationen unter Workbench und CLI
- BASIC: Laden, Speichern, sequenzielle und relative Dateien
- DOS-Funktionen
- Fileverwaltung: Blocktypen, Boot-Block, Checksummen, File-Header, Hash-Berechnung, Bitmap
- Viren: Boot-Block und Schutz
- Trackdisk-Device: Befehle, Strukturen, Nachrichten
- Trackdisk-Task: Funktion und Aufbau
- Diskettenzugriff ohne DOS: MFM, GCR, Aufbau von Tracks, Block-Header, Datenblock, Prüfsummen, Codierung und Decodierung, Hardware-Register, SYNC, Interrupts
- Diskmonitor
- Fast-Copy: Komplette Diskette in ca. 1 Minute kopieren
- Crunch-Copy: Durch Packroutinen weniger Diskettenwechsel
- Deep-Copy: Kopiert praktisch jeden Kopierschutz, ST- und PC-Disketten
- Floppyspeeder: Superschnelle Trackdisk-Routinen beschleunigen den Diskettenzugriff

Abraham, Gelfand, Grote
Das große Amiga-Floppybuch
398 Seiten, DM 59,–
ISBN 3-89011-180-7

Bücher zum Amiga

Von der Hardware über den Betriebssystemkern EXEC bis zum DOS finden Sie in diesem Buch die entscheidenden Informationen zum Amiga. Und zwar so verständlich, daß auch Nicht-Profis die Arbeitsweise des Amiga-Betriebssystems schnell verstehen werden.

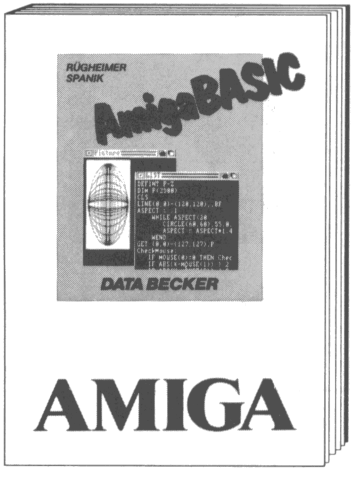
Aus dem Inhalt:

- die Chips des Amiga (68000, CIA, Agnus, Denise, Paula)
- Die Schnittstellen (Video, Audio, RGB, Centronics, seriell, Floppy, Gameport, Expansionport, Tastatur)
- Programmierung der Hardware (Speicherbelegung, Interrupts, Copper, Blitter, Diskcontroller)
- Strukturen des EXEC (Node, List, Libraries, Tasks)
- Funktion des Multitasking (Task-switching, Kommunikation zwischen Tasks, Exceptions, Traps, Speicherverwaltung)
- I/O-Handhabung beim Amiga durch Devices und I/O-Request)
- Interrupt-Handhabung und Verwaltung der Ressources
- RESET-feste Programme und Strukturen, Dokumentation der RESET-Routine
- EXEC-Base (Dokumentation und Nutzung der Systemvariablen)
- DOS-Bibliothek (Funktionen, Parameter, Fehlermeldungen)
- Disketten (Bootvorgang, Datenstrukturen, Programmaufbau, IFF-Format)
- Programmstart, Parameter, Aufruf von CLI und Workbench, detaillierte Beschreibung des internen Aufbaus der CLI-Befehle (interne DOS-Bibliothek)
- Ein-/Ausgaben (Tastatur, Bildschirm, Diskette, parallele und serielle Schnittstelle)
- Devices (Trackdisk, Console, Narrator, SER, PAR, PRT, Gameport)

Dittrich, Gelfand, Schemmel
AMIGA Intern
Hardcover, 640 Seiten, DM 69,–
ISBN 3-89011-104-1

Bücher zum Amiga

AmigaBASIC für alle. Im ersten Teil werden Sie Schritt für Schritt – und vor allem auf verständliche Art und Weise – in die Programmierung des AMIGA eingeführt: Grafik und Sound gehören genauso dazu wie Datenverwaltung und Statistik. Im zweiten Teil finden Sie alle gelernten Befehle mit Syntax- und Parameterangaben zum schnellen Nachschlagen. Dazu gibt es Programme und Utilities in Hülle und Fülle.

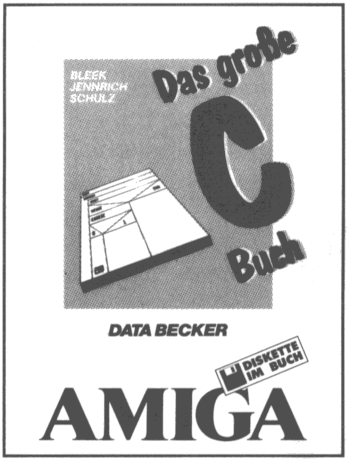


Aus dem Inhalt:

- Das Videotitel-Programm zeigt die OBJECT-Animation
- Das Balken- und Tortengrafik-Programm erklärt die Grafikbefehle
- Das Malprogramm mit Windows, Pulldowns, Mausbefehlen, Füllmustern, Einlesen und Abspeichern von IFF-Bildern
- Das Statistikdaten-Programm hilft, sequentielle Dateien zu verstehen
- Die Datenbank zeigt den Umgang mit relativen Dateien
- Das Sprachutility sorgt für mehr Verständnis bei der Sprachprogrammierung
- Das Synthesizer-Programm führt Sie in die Welt der Töne, Wellenformen und Hüllkurven

Rügheimer, Spanik
AmigaBASIC
Hardcover, 775 Seiten, DM 59,–
ISBN 3-89011-209-9

Wer den Einstieg in C geschafft hat, wird sich so manches Mal fragen, warum das gut durchdachte Programm nicht funktioniert. Hier helfen fundierte Informationen von Profis weiter: Funktionsweise von Compiler Assembler und Linker (am Beispiel von Aztec 3.4.), komfortable Oberfläche für eigene Programme mit Intuition und ein großes Projekt professionell programmiert - mit diesem Buch bleibt keine Frage offen.



Aus dem Inhalt:

- Funktionsweise des Aztec-Compilers (Compiler, Assembler, Linker)
- Wie funktionieren INCLUDE, DEFINE und CASTS?
- Debugging und Optimierung des Assembler-Sources
- Knifflige Programmierprobleme: Sprungtabellen und dynamische Arrays in C
- Einbinden von Assembler-Source in den C-Source
- Alles über Intuition-Programmierung (Windows, Screens, Pull-Down-Menüs, Requester, Gadgets)
- Programmierung des Console-Device
- Ein professionelles Editor zeigt alle Probleme bei der Erstellung größerer Programme
- Richtiger Einsatz von MAKE
- Debugging von C-Programmen mit verschiedenen Hilfsmitteln
- Ein kompletter Text-Editor als Source in mehreren Entwicklungsstufen
- Folding-Technik (Der Editor kann Textteile und Funktionen wegfalten, das erhöht die Übersichtlichkeit enorm.)

Bleek, Jennrich, Schulz
Das große C-Buch
Hardcover, 682 Seiten, DM 69,-
ISBN 3-89011-191-2

Schreiben Sie Ihre Programme in Maschinenesprache – und Sie werden sehen, wie schnell ein Amiga sein kann. Das nötige Know-how liefert Ihnen dieses Buch: Grundlagen des 68000, das Amiga-Betriebssystem, Druckeransteuerung, Diskettenoperationen, Sprachausgabe, Windows, Screens, Register, Pull-Down-Menüs . . . Aber es wird auch gleich gezeigt, wie man mit den wichtigsten Assemblern arbeitet.

Dittrich
Amiga Maschinensprache
Hardcover, 288 Seiten, DM 49,–
ISBN 3-89011-076-2

DAS STEHT DRIN:

Amiga-Anwender, die nur mit Standard-Software arbeiten, werden die komfortable Workbench kaum verlassen. Wer jedoch mehr aus seinem Rechner herausholen will, macht dies über das AmigaDOS. Daß man dabei nicht unbedingt ein Profi sein muß, zeigt dieses Buch. Denn angefangen mit dem Einsteigerteil bis zum internen Aufbau erfährt man Schritt für Schritt alles, was bei der praktischen Arbeit auch mit dem neuen Betriebssystem V1.3 des AmigaDOS zu beachten ist. Zudem sorgt ein gut strukturierter, umfassender Nachschlageteil dafür, daß auch später jede benötigte Information schnell wiedergefunden werden kann.

Aus dem Inhalt:

- Aufgabe und Handhabung des AmigaDOS
- Ausführliche Erläuterung der CLI/Shell und der neuen CLI-Befehle
- Die beiden DOS-Editoren ED und EDIT
- Tips und Tricks (Joker, Umlenkung der Ein-/Ausgabe)
- Die Devices und deren Nutzung vom CLI/Shell aus
- Batch-Dateien: Wie werden sie erstellt, genutzt . . .
- Optimale Nutzung der Multitasking-Fähigkeiten des AmigaDOS
- Wie sind die CLI-Befehle aufgebaut
- Wie werden eigene CLI-Befehle erstellt
- Setzen Sie Prioritäten mit dem neuen TaskPri
- Zu schnell? Stufenloses Abbremsen mit Delay
- Vom CLI/Shell aus eine beliebige Zeichenkette in einem Text durch eine andere ersetzen

UND GESCHRIEBEN HABEN DIESES BUCH:

Rüdiger Kerkloh und Manfred Tornsdorf liefern die Grundlage des Buches, denn beide nutzen seit Jahren den Amiga bei ihrer täglichen Arbeit, Rüdiger Kerkloh bei seinem Studium und Manfred Tornsdorf als Amiga-Lektor bei DATA BECKER. Abgerundet wird der Inhalt des Buches durch die Erfahrungen, die Bernd Zoller während seines Informatik-Studiums und bei seiner Arbeit als Lektoratsassistent gesammelt hat.

ISB N 3-89011-306-0 DM +059.00

DM 59,–
ÖS 460,–
sFr 57,–

DATA BECKER

05900

9 783890 113067

This was brought to you

from the archives of

http://retro-commodore.eu